A MARSHALL CAVENDISH **24** COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol. 2 No 24

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Steve Bielschowsky/Bernard Fallon. Pages 733, 734, 738, Paul Chave/Ian Stephen/Zefa. Pages 740, 742, 746, Mohsen John Modaberi. Page 748, Steve Bielschowsky/Bernard Fallon. Pages 753, 754, 755, Russell Walker. Pages 757, 764, Peter Reilly. Pages 758, 760, 762, Andrew MacConville.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

GET OFF TO A FLYING START

- FLYING THE AEROPLANE ON AUTO-PILOT
- NEARING THE RUNWAY
- PLOTTING THE COURSE
- WORKING DISPLAYS



Start the engines in part two of the flight simulator, and see your instrument panel come alive. But hang on to your hats, because your auto-pilot has gone crazy!

In the first part of this article, you entered the lines that recreated the interior of a cockpit on your screen. And in the case of the Dragon and the Tandy, you also set your aeroplane high in the sky, motionless but ready.

In this part, the aeroplane is set in motion and the instrument panel comes to life, so that although you are not yet in control, you can see how the aeroplane's instrument panel responds to the movements of the craft.

FLYING THE AEROPLANE

This is by far the longest part of the program. A complex series of interdependent variables have to be updated constantly to control the progress of the aeroplane. At the same time, the instrument panel needs to be redrawn as the position and height of the aeroplane change, and the dials monitor the movement.

NEARING THE RUNWAY

A radar image of the runway shows you the angle at which you are approaching it on the Spectrum, Acorn and Commodore computers. On the Dragon and Tandy, an image of the runway itself can be seen through the window as you approach close enough to land. The runway 'grows' progressively larger as you near the ground, using the computers' ability to draw ellipses.

PLOTTING THE COURSE

Many factors have to be taken into account before you can plot the position of the aeroplane accurately. The direction in which you are flying, for instance, is affected by the wind direction, and the roll of the aeroplane. The speed at which you travel forwards depends partly on the wind speed. The distance you fall, or climb, is connected to the speed at which you are flying, and so on.

To update the dials and counters, the changing variables must be assessed according to how they affect the readings: then they can be redrawn.

```
2 LET WY=0: LET WX=0: LET GZ=0: LET
   GY=0: LET GX=0
5 LET RW=0: LET Y1=120: LET Y2=120:
   LET Y3=40: LET Y4=40: LET POW=0:
   LET GC=0: LET RB=0: LET LL=0: LET
   YC=0: LET AD=0: LET ST=0: LET
   RL=0: LET BC=0: LET NC=0: LET
   PT=0: LET PX=0: LET VZ=0: LET
   VY=0: LET VX=0
```

```
500 LET RA=AD*C: LET VX=AS*SIN RA
510 LET VY=AS*COS RA: RETURN
1000 LET PZ=PZ+GZ: LET PY=PY+GY:
   LET PX=PX+GX
1025 IF ST=1 THEN PRINT OVER 1;AT
   4,12;"S□T□A□L□L": LET ST=Ø:
   GOTO 1Ø4Ø
1Ø3Ø IF AS<3Ø THEN GOSUB 15ØØ
1Ø4Ø LET AD=AD+RL: IF AD<Ø THEN LET
   AD=AD+36Ø
1Ø5Ø IF AD>359 THEN LET AD=AD-36Ø
1Ø6Ø LET VZ=AS*SIN (PT*C)-1Ø+AS/15
1Ø7Ø LET GZ=VZ: LET GY=VY+WY: LET
   GX=VX+WX
1Ø8Ø IF VY=Ø THEN LET GD=-PI/2:
   GOTO 11ØØ
1Ø9Ø LET GD=-ATN (VX/VY)/C
11ØØ GOSUB 5ØØ
111Ø RETURN
15ØØ LET ST=1: PRINT OVER 1;AT
   4,12;"S□T□A□L□L": FOR M=1 TO
   4: FOR N=2Ø TO -2Ø STEP -4: BEEP
   .Ø1,N: NEXT N: NEXT M
151Ø LET RL=INT (RND*21)-9: LET
   PT=-21-INT (RND*5)
152Ø RETURN
218Ø IF GC<>Ø THEN GOSUB 22ØØ
219Ø LET AS=AS+16*(TC*3Ø-AS-
   8*PT)/AS: GOSUB 22ØØ: GOTO 22Ø5
22ØØ PLOT 35,5Ø: DRAW OVER 1;15*SIN
   (AS*PI/2ØØ),15*COS (AS*PI/2ØØ): RETURN
22Ø5 IF GC<>Ø THEN PLOT 155,5Ø: DRAW
   OVER 1;1Ø*SIN (TN*PI/5),1Ø*COS
   (TN*PI/5): PLOT 155,5Ø: DRAW OVER
   1;15*SIN (UN*PI/5ØØ),15*COS
   (UN*PI/5ØØ)
221Ø LET TN=PZ/1ØØØ: LET UN=PZ-
   1ØØØ*INT TN: PLOT 155,5Ø: DRAW OVER
   1;1Ø*SIN (TN*PI/5),1Ø*COS (TN*PI/5):
   PLOT 155,5Ø: DRAW OVER 1;15*SIN
   (UN*PI/5ØØ),15*COS (UN*PI/5ØØ)
222Ø IF GC<>Ø THEN GOSUB 223Ø
2225 IF POW=-1 AND TC>.2 THEN LET
   TC=TC-.2
2226 IF POW=1 AND TC<8.8 THEN LET
   TC=TC+.2
2228 GOSUB 223Ø: GOTO 224Ø
223Ø PLOT 215,5Ø: DRAW OVER 1;15*SIN
   (TC*PI/5),15*COS (TC*PI/5): RETURN
224Ø PRINT AT 21,2;ABS INT AD;"□□"
225Ø IF PY=Ø THEN LET RB=Ø: GOTO 226Ø
2255 LET RB=ATN (PX/PY)/C: IF PY>Ø
   THEN LET RB=RB+18Ø
226Ø IF RB<Ø THEN LET RB=RB+36Ø
227Ø PRINT AT 21,1Ø;INT RB;"□□";
   AT 21,18;ABS INT PX;"□□"
228Ø PRINT AT 21,25;INT (SQR
   (PY*PY+PX*PX));"□"
229Ø IF (Y1<=11Ø AND Y2<=11Ø) OR
   (Y1>=13Ø AND Y2>=13Ø) THEN
   GOTO 23ØØ
2295 IF GC<>Ø THEN PLOT OVER
   1;X1,168-Y1: DRAW OVER 1;X2-PEEK
   23677,168-Y2-PEEK 23678
23ØØ LET YC=12Ø+(PT/3): LET X1=8Ø:
   LET X2=11Ø: LET Y1=YC+17*TAN
   (RL*2*C): LET Y2=YC-17*TAN (RL*2*C)
231Ø IF (YC<11Ø OR YC>13Ø) AND RL=Ø
   THEN GOTO 2376
232Ø IF Y1<11Ø THEN LET X1=95-
   (95-X1)*(11Ø-YC)/(Y1-YC): LET
   Y1=11Ø: GOTO 234Ø
233Ø IF Y1>13Ø THEN LET X1=95-
   (95-X1)*(13Ø-YC)/(Y1-YC): LET
   Y1=13Ø
234Ø IF Y2<11Ø THEN LET X2=95-
   (95-X2)*(11Ø-YC)/(Y2-YC): LET
   Y2=11Ø: GOTO 236Ø
235Ø IF Y2>13Ø THEN LET X2=95-
   (95-X2)*(13Ø-YC)/(Y2-YC): LET
   Y2=13Ø
236Ø IF X1<8Ø OR X2>11Ø THEN GOTO
   2376
237Ø PLOT OVER 1;X1,168-Y1: DRAW OVER
   1;X2-PEEK 23677,168-Y2-PEEK 23678
2376 IF (RL=RR AND PP=PT) THEN GOTO
   25ØØ
2377 IF (Y3<=2 AND Y4<=2) OR (Y3>
   =9Ø AND Y4>=9Ø) THEN GOTO 238Ø
2378 IF GC<>Ø THEN PLOT OVER
   1;X3,176-Y3: DRAW OVER 1;X4-PEEK
   23677,(176-Y4)-PEEK 23678
238Ø LET YC=33+PT*4: LET X3=11: LET
   X4=244: LET Y3=YC+118*TAN
   (RL*2*C): LET Y4=YC-118*TAN
   (RL*2*C)
239Ø IF (YC<2 OR YC>9Ø) AND RL=Ø
   THEN GOTO 245Ø
24ØØ IF Y3<2 THEN LET X3=128-
   (128-X3)*(2-YC)/(Y3-YC):
   LET Y3=2: GOTO 242Ø
241Ø IF Y3>9Ø THEN LET X3=128-
   (128-X3)*(9Ø-YC)/(Y3-YC):
   LET Y3=9Ø
242Ø IF Y4<2 THEN LET X4=128-
   (128-X4)*(2-YC)/(Y4-YC):
   LET Y4=2: GOTO 244Ø
243Ø IF Y4>9Ø THEN LET X4=128-
   (128-X4)*(9Ø-YC)/(Y4-YC): LET
   Y4=9Ø
244Ø IF X3<11 OR X4>244 THEN GOTO
   25ØØ
2445 OVER 1: PLOT X3,176-Y3: DRAW
   X4-PEEK 23677,(176-Y4)-PEEK
   23678: OVER Ø
25ØØ GOSUB 8ØØØ
25Ø5 IF GC=Ø THEN LET GC=1
251Ø LET RR=RL: LET PP=PT: RETURN
5Ø8Ø LET GZ=VZ: LET GY=VY+WY: LET
   GX=VX+WX
5Ø9Ø LET TC=5
51ØØ LET RT=3: LET TP=5: LET WR=5Ø
55ØØ IF INT (RND*5)=1 THEN LET
   RL=RL+INT (RND*5)-2: IF INT
   (RND*5)=1 THEN LET PT=PT+3-INT
   (RND*2)+1*2
551Ø GOSUB 1ØØØ: IF PZ<Ø THEN GOTO
   553Ø
552Ø GOSUB 218Ø: GOTO 55ØØ
553Ø GOTO 55ØØ
8ØØØ IF GC<>Ø THEN PLOT 127,174:
   DRAW OVER 1;OX,OY
8Ø1Ø LET OX=16*SIN (RB*(PI/18Ø)): LET
   OY=-(16*ABS COS (RB*(PI/18Ø)))
8Ø2Ø PLOT 127,174: DRAW OVER 1;OX,OY
8Ø25 LET WB=AD: IF AD>18Ø THEN LET
   WB=WB-36Ø
8Ø26 IF RB>18Ø THEN LET
   WB=WB+36Ø-RB: GOTO 8Ø4Ø
8Ø3Ø LET WB=WB-RB
8Ø4Ø IF RW=1 THEN PLOT OVER
   1;RDX,175-RDY
8Ø5Ø LET RW=Ø: IF ABS WB>57 THEN
   RETURN
8Ø6Ø LET RDX=X3+INT
   (((X4-X3)/2)-SIN
   (WB*(PI/18Ø))*(X4-X3)*.6)
8Ø7Ø LET RDY=Y3+((Y4-Y3)*
   ((RDX-X3)/(X4-X3))+2)
8Ø8Ø IF RDY<2 OR RDY>9Ø OR RDX<11
   OR RDX>244 THEN RETURN
8Ø9Ø LET RW=1: PLOT OVER
   1;RDX,175-RDY
81ØØ RETURN
```



The POKE in Line 1 sets the computer to print in upper case letters. Line 5 sets all the variables to 0. Line 110, which you entered last time, sends the program to 5000 and draws the cockpit of the aeroplane, then Line 5080 sets the variables that control the position of the aeroplane in the sky: GZ refers to the distance the aeroplane is along the Z axis—its height up or down; VZ is the velocity along the same axis; VY refers to the velocity of the plane forwards or backwards—that is, along the Y axis; WY is the windspeed in the same direction. GX, VX and WX correspond to the distance moved, velocity and windspeed along the X axis, which runs right to left.

Line 5090 sets the rev counter, and Line 5100 defines the limits of the roll (RT), the pitch (TP) and the width of the runway (WR).

Line 5500 is the temporary command standing in for the main control routine, which you will enter in the next part.

Line 5510 sends you to the subroutine that begins at Line 1000 and ends at Line 1110. This subroutine updates all the variables as the aeroplane flies. Lines 1025 and 1030 check to see whether you have allowed your aeroplane to stall by letting its speed drop below 30 metres per second. If so, it sends you to the subroutine 1500 to 1520, which recreates the effect of a stall. Line 1510 makes sure that you won't know quite what will happen when you go into your stall—you could just drop straight out of the sky, or spin wildly as you fall.

The subroutine to which you are sent by Line 1100, contained in Lines 500 and 510, works out the angle at which you are flying.

The next major subroutine, starting at Line 2180, and ending at Line 2510, redraws the dials and counters when the information they record is updated. Line 2180 checks the GC Counter to see if there is an image that needs to be replaced. The subroutine contained in the Line 2200 draws the new position of the hand of the airspeed dial. Lines 2205 and 2210 calculate and redraw the new position of both hands on the altitude dial. Line 2230 moves the hand on the new counter using the calculations made in Lines 2225 and 2226.

The rev counter is updated by Line 2240. Lines 2250 to 2270 calculate the new runway bearing and drift, while the distance is calculated and printed by Line 2280.

The artificial horizon in the second dial is calculated, and is drawn, by Lines 2280 to 2370.

The line of the actual horizon is checked by Line 2376, and Lines 2377 to 2445 calculate and redraw it if it can be seen through the cockpit window.

The subroutine that starts at 8000 and ends at 8100, to which you are sent by Line 2500, calculates and draws the radar image of the runway, which you see at the top of your screen, and also the dot of the runway that appears just below the horizon line through the cockpit window, when it is in view.



```
10 GOTO 5000
500 RA=AD*C:VX=AS*SIN(RA)
510 VY=AS*COS(RA):RETURN
1000 PZ=PZ+GZ:PY=PY+GY:
   PX=PX+GX
1010 PT=PT+NC:RL=RL+BC
1020 AS=AS+16*(TC*30-AS-8*PT)/AS
1030 IF SL=1 THEN TEXT 60,50,
   "STALL",0,3,8:SL=0:GOTO1050
1040 IF AS<30 THEN GOSUB 1500
1050 AD=AD+RL:IF AD<0 THEN
   AD=AD+360
1060 IF AD>359 THEN AD=AD-360
1070 VZ=AS*SIN(PT*C)-10+AS/15
1080 GZ=VZ:GY=VY+WY:GX=VX+WX
1090 IF VY=0 THEN GD=-π/2:GOTO
   1110
1100 GD=-ATN(VX/VY)/C
1110 GOSUB 500
1120 RETURN
1500 SL=1:TEXT 60,50,"STALL",1,3,8
1510 RL=INT(RND(1)*21)-9:PT=
   21-INT(RND(1)*5)
1520 RETURN
2000 LINE 20,150,20+13*SIN(AS*π/200),
   150-13*COS(AS*π/200),4
2010 TN=PZ/1000:UN=PZ-1000*
   INT(TN)
2020 LINE 100,150,100+6*SIN(TN*π/5),
   150-6*COS(TN*π/5),4
2030 LINE 100,150,100+13*SIN(UN*
   π/500),150-13*COS(UN*π/500),4
2040 LINE 140,150,140+13*SIN(TC*
   π/5),150-13*COS(TC*π/5),4
2050 TEXT 0,190,STR$(ABS(INT(AD))),
   4,1,7:RETURN
2060 IF PY=0 THEN RB=0
2065 IF PY<>0 THEN RB=ATN
   (PX/PY)/C:IF PY>0 THEN RB=RB+180
2070 IF RB<0 THEN RB=RB+360
2075 GOSUB 7000
2080 TEXT 35,190,STR$(INT(RB)),
   1,1,7:TEXT 70,190,STR$
   (ABS(INT(PX))),1,1,7
2090 TEXT 110,190,STR$(INT(SQR
   (PY*PY+PX*PX))),1,1,7
2095 S1=INT(RB):S2=ABS(INT(PX))
2096 S3=INT(SQR(PY*PY+PX*PX))
2098 IF KJ=1 THEN LINE X1,Y1,X2,Y2,4
2100 KJ=0:YC=150+(PT/3):X1=50:
   X2=70:Y1=YC+17*TAN(RL*2*C):
   Y2=YC-17*TAN(RL*2*C)
2110 IF (YC<137 OR YC>163) AND RL=0
   THEN 2320
2120 IF Y1<137 THEN X1=60-
```

```
   (6Ø-X1)*(14Ø-YC)/(Y1-YC):
   Y1=14Ø:GOTO 214Ø
213Ø IF Y1>163 THEN X1=6Ø-(6Ø-X1)
   *(16Ø-YC)/(Y1-YC):Y1=16Ø
214Ø IF Y2<137 THEN X2=6Ø-
   (6Ø-X2)*(14Ø-YC)/(Y2-YC):
   Y2=14Ø:GOTO 216Ø
215Ø IF Y2>163 THEN X2=6Ø-(6Ø-X2)
   *(16Ø-YC)/(Y2-YC):Y2=16Ø
216Ø IF X1<5Ø OR X2>7Ø THEN 219Ø
217Ø LINE X1,Y1,X2,Y2,4:KJ=1
219Ø IF RL=RR AND PP=PT THEN 229Ø
22ØØ IF HF=1 THEN LINE X3,Y3,X4,Y4,Ø
221Ø HF=Ø:YC=33+PT*4:X3=Ø:X4=
   159:Y3=YC+59*TAN(RL*2*C):
   Y4=YC-59*TAN(RL*2*C)
222Ø IF (YC<Ø OR YC>1Ø9) AND RL=Ø
   THEN 229Ø
223Ø IF Y3<Ø THEN X3=8Ø-(8Ø-X3)*
   (-YC)/(Y3-YC):Y3=Ø:GOTO 225Ø
224Ø IF Y3>1Ø9 THEN X3=8Ø-(8Ø-X3)*
    (1Ø9-YC)/(Y3-YC):Y3=1Ø9
225Ø IF Y4<Ø THEN X4=8Ø-(8Ø-X4)*
   (-YC)/(Y4-YC):Y4=Ø:GOTO 227Ø
226Ø IF Y4>1Ø9 THEN X4=8Ø-(8Ø-X4)*
   (1Ø9-YC)/(Y4-YC):Y4=1Ø9
227Ø IF X3<Ø OR X4>159 THEN 229Ø
228Ø HF=1:LINE X3,Y3,X4,Y4,3
229Ø WB=AD:IF AD>18Ø THEN
   WB=WB-36Ø
23ØØ IF RB>18Ø THEN WB=WB+
   36Ø-RB:GOTO 231Ø
23Ø5 WB=WB-RB
231Ø IF ABS(WB)>6Ø AND ABS
   (PY)>1ØØØ THEN 235Ø
232Ø AN=59/(6Ø*SQR((X3-X4)*(X3-
   X4)+(Y3-Y4)*(Y3-Y4)))
2325 X5=(X3+X4)/2+SGN(X3-
   X4)+WB*AN*(X3+X4)
233Ø Y5=(Y3+Y4)/2+2+WB*AN*
   (Y3-Y4)
2335 IF X5<Ø OR X5>159 OR Y5<Ø OR
   Y5>1Ø9 THEN 235Ø
234Ø IF ABS(PY)<1ØØØ THEN
   R=8-Y5/1Ø: GOTO 235Ø
2345 R=4ØØØ/ABS(PY): IF R*1Ø+Y5>8Ø
   THEN R=8-Y5/1Ø
235Ø GOSUB 8ØØØ
237Ø RR=RL:PP=RT:RETURN
5ØØØ PRINT "■■":NRM:COLOUR
   6,6:PP=-1:RR=-1
5Ø1Ø C=π/18Ø:PY=-2ØØØØ:PZ=2ØØØ:
   AS=15Ø
5Ø2Ø PRINT "□INPUT WIND SPEED
   (■1-5Ø■) M/S"
5Ø25 PRINT "AND DIRECTION
   (■Ø-359■) DEGREES■":FLASH
   5,1Ø
5Ø3Ø INPUT XØ,X1:IF XØ>5Ø OR XØ<1 OR
   X1<Ø OR X1>359 THEN 5ØØØ
5Ø4Ø XØ=XØ/3:OFF:POKE 65Ø,128
5Ø5Ø PRINT AT(Ø,2Ø)"WIND SPEED
   □=";3*XØ;"M/S":PRINT
   "□DIRECTION□=";X1;
   "DEGREES"
5Ø6Ø WY=-XØ*COS(X1*C)
5Ø7Ø WX=-XØ*SIN(X1*C)
5Ø8Ø GZ=VZ:GY=VY+WY:
   GX=VX+WX
5Ø9Ø TC=5:RT=3:TP=5:WR=5Ø:
   PAUSE 2
55ØØ GOSUB 2ØØØ: IF INT(RND(1)*1Ø)=1
   THENRL=RL+SGN(TL)*INT
   (RND(1)*4)-1
551Ø IF INT(RND(1)*1Ø)=1 THEN
   PT=PT+3-INT(RND(1)*4+1)*2
552Ø GOSUB 1ØØØ:IF PZ<=Ø THEN 554Ø
553Ø GOSUB 2ØØØ:GOSUB 2Ø6Ø:GOTO 55ØØ
554Ø GOTO 554Ø
7ØØØ TEXT 35,19Ø,STR$(S1),2,1,7:
   TEXT 7Ø,19Ø,STR$(S2),2,1,7
7Ø1Ø TEXT 11Ø,19Ø,STR$(S3),2,1,7:
   RETURN
8ØØØ IF WQ=1 THEN LINE 78,Ø,OX,OY,Ø
8Ø1Ø WQ=1:OX=78-(16*SIN(RB*
   (π/18Ø))):OY=(16*ABS(COS
   (RB*(π/18Ø))))
8Ø2Ø LINE 78,Ø,OX,OY,2
8Ø25 WB=AD:IF AD>18Ø THEN
   WB=WB-36Ø
8Ø26 IF RB>18Ø THEN WB=WB+
   36Ø-RB:GOTO8Ø4Ø
8Ø3Ø WB=WB-RB
8Ø4Ø IF RW=1 THEN PLOT G1,G2,Ø
8Ø5Ø RW=Ø:IF ABS(WB)>57 THEN
   RETURN
8Ø6Ø RX=X3+INT(((X4-X3)/2)-
   SIN(WB*(π/18Ø))*(X4-X3)*.6)
8Ø7Ø RY=Y3+((Y4-Y3)*((RX-
   X3)/(X4-X3))+2)
8Ø8Ø IF RY<Ø OR RY>1Ø9 OR RX<Ø OR
   RX>159 THEN RETURN
8Ø9Ø RW=1:PLOT RX,RY,2:G1=RX:G2=RY
81ØØ RETURN
```

The program jumps to Line 5ØØØ as soon as it is RUN. Lines 5ØØØ and 5Ø1Ø initialize a range of variables which control the position of the aeroplane in the sky. Initially, the aircraft is set 20,000 metres from the runway, and 2000 metres from the ground.

Lines 5Ø2Ø to 5Ø3Ø allow the pilot to choose the wind strength and direction, and check if the inputs are within the permitted range. The values are PRINTed in Line 5Ø5Ø, before Lines 5Ø6Ø and 5Ø7Ø calculate the wind speed in the forwards direction and the wind speed in the left to right direction. WX and WY are then used in Line 5Ø8Ø to adjust the position of the aeroplane.

Line 5Ø9Ø sets the rev counter (TC), the limits of the roll (RT) and the pitch (TP), and the runway width (WR).

Line 55ØØ calls the subroutine starting at Line 2ØØØ, and ending at Line 2Ø5Ø. The subroutine updates the dials in the cockpit display. Lines 2ØØØ, 2Ø2Ø, 2Ø3Ø and 2Ø4Ø draw each of the needles in turn, according to the speed and position of the aeroplane.

RETURNing to Line 55ØØ, the remainder of the line and the whole of Line 551Ø act as a stand-in for the control routine which you'll be adding in the next part of this article.

Line 552Ø calls the subroutine starting at Line 1ØØØ. The subroutine updates all the variables as the aeroplane flies. Lines 1Ø3Ø and 1Ø4Ø checks for stalls—if you have allowed the aeroplane to drop below 30 metres per second. If the airspeed has dropped, the program jumps to the subroutine starting at Line 15ØØ, which displays the stall message on screen, and sets the roll and pitch randomly to simulate losing control of the aeroplane. Line 111Ø calls the subroutine at Line 5ØØ. Lines 5ØØ and 51Ø simply work out the speed of ascent or descent and the speed from left to right.

RETURNing from Line 112Ø to Line 552Ø, the program checks if the aeroplane has touched down, and jumps to Line 554Ø if it has. Line 554Ø is just a stop put here temporarily.

Line 553Ø calls the subroutine at Line 2ØØØ to update the dials. Next, the subroutine at Line 2Ø6Ø is called. Lines 2Ø6Ø to 2Ø7Ø update the runway bearing before the last three numerical displays are deleted by the subroutine at Line 7ØØØ onwards. The new readings are displayed by Lines 2Ø8Ø and 2Ø9Ø. New values for S1, S2 and S3 are calculated in Lines 2Ø95 to 2Ø96—these are used in Lines 7ØØØ and 7Ø1Ø.

The artificial horizon is drawn by the LINE in Line 2Ø98—the corresponding erasing statement is at Line 217Ø and KJ is simply a flag so that the computer knows when to draw a new artificial horizon. Lines 21ØØ to 216Ø update the control variables. The lines from 22ØØ to 228Ø deal with the actual horizon in exactly the same way.

The new position of the runway is calculated by Lines 229Ø to 237Ø, using the subroutine between Lines 8ØØØ and 81ØØ to draw the runway in the appropriate position.

After the horizon has been redrawn, the display has been fully updated, and Line 553Ø sends the program back to Line 55ØØ. Notice how the main loop of the program is just these four lines.

```
15 GOTO 115Ø
38Ø DEF PROCWORKOUT
```

```
390 AS2=AS:TN2=TN:UN2=UN
400 PZ=PZ+GZ:PY=PY+GY:
    PX=PX+GX
410 AS=AS+16*(TC*30-AS-
    8*PT)/AS
420 IF ST=1 THEN PRINTTAB (15,10)
    "□□□□□":ST=0:GOTO 440
430 IF AS<30 THEN GOSUB 520
440 AD=AD+RL:IF AD<0 THEN
    AD=AD+360
450 IF AD>359 THEN AD=AD-360
460 VZ=AS*SIN(PT*C)-10+AS/15
470 GZ=VZ:GY=VY+WY:GX=VX+WX
480 IF VY=0 THEN GD=-PI/2 ELSE
    GD=-ATN(VX/VY)/C
490 RA=AD*C:VX=AS*SIN(RA)
500 VY=AS*COS(RA)
510 ENDPROC
520 ST=1:PRINTTAB(15,10)"STALL"
530 RL=RND(21)-10:PT=-20-RND(5)
540 RETURN
550 DEF PROCINIT
560 MOVE190,250:DRAW 190+SIN
    (AS*PI/200)*80,250+COS
    (AS*PI/200)*80:TN=PZ/1000:
    UN=PZ-1000*INT(TN):
    MOVE790,250:DRAW 790+SIN
    (TN*PI/5)*40,250+COS
    (TN*PI/5)*40
570 MOVE790,250:DRAW 790+SIN
    (UN*PI/500)*80,250+COS
    (UN*PI/500)*80:MOVE1090,250:
    DRAW 1090+SIN(TC*PI/5)*80,
    250+COS(TC*PI/5)*80
580 ENDPROC
590 DEF PROCINFO
600 MOVE190,250:DRAW 190+SIN
    (AS2*PI/200)*80,250+COS
    (AS2*PI/200)*80
610 MOVE190,250:DRAW 190+SIN
    (AS*PI/200)*80,250+COS
    (AS*PI/200)*80
620 MOVE790,250:DRAW 790+SIN
    (TN2*PI/5)*40,250+COS
    (TN2*PI/5)*40
630 TN=PZ/1000:UN=PZ-1000*
    INT(TN):MOVE790,250:DRAW
    790+SIN(TN*PI/5)*40,250+
    COS(TN*PI/5)*40
640 MOVE790,250:DRAW 790+SIN
    (UN2*PI/500)*80,250+COS
    (UN2*PI/500)*80
650 MOVE790,250:DRAW 790+SIN
    (UN*PI/500)*80,250+COS
    (UN*PI/500)*80
660 MOVE1090,250:DRAW 1090+SIN
    (TC2*PI/5)*80,250+COS
    (TC2*PI/5)*80
670 MOVE1090,250:DRAW 1090+SIN
    (TC*PI/5)*80,250+COS
    (TC*PI/5)*80
680 IF PY=0 THEN RB=0 ELSE
    RB=ATN(PX/PY)/C:IF PY>0
    THEN RB=RB+180
690 IF RB<0 THEN RB=RB+360
700 PRINTTAB(4,30);INT(AD)
    "□□";TAB(14,30);INT(RB)
    "□□";TAB(23,30);INT(PX)
    "□□□□";TAB(33,30);
    INT(SQR(PX*PX+PY*PY));"□"
710 RB2=RB:AD2=AD:IF RB>290 AND
    AD<70 THEN AD2=AD+360
720 IF AD>290 AND RB<70 THEN
    RB2=RB+360
730 IF HF3=1 THEN MOVERX-DRX,
    968-DRY:DRAWRX+DRX,
    968+DRY
740 HF3=0:IF ABS(RB2-AD2)>90 THEN
    780
750 RX=TAN(RAD(AD-RB))*250+
    640:IF RX<4 OR RX>1276 THEN 780
760 DRX=50*SIN(RAD(RB)):DRY=
    50*COS(RAD(RB))
770 HF3=1:MOVERX-DRX,968-DRY:
    DRAWRX+DRX,968+DRY
780 IF HF2=1 THEN MOVEX1,Y1:
    DRAWX2,Y2
790 HF2=0:YC=250-PT*4:X1=420:
    X2=560:Y1=YC+29*TAN(RL*2*C):
    Y2=YC-29*TAN(RL*2*C)
800 IF (YC<180 OR YC>320) AND RL=0
    THEN 1040
810 IF Y1>320 THEN X1=490-
    (490-X1)*(320-YC)/(Y1-YC):
    Y1=320:GOTO 830
820 IF Y1<180 THEN X1=490-
    (490-X1)*(180-YC)/(Y1-YC):
    Y1=180
830 IF Y2>320 THEN X2=490-
    (490-X2)*(320-YC)/(Y2-YC):
    Y2=320:GOTO 850
840 IF Y2<180 THEN X2=490-
    (490-X2)*(180-YC)/(Y2-YC):
    Y2=180
850 IF X1<420 OR X2>560 THEN 870
860 HF2=1:MOVEX1,Y1:DRAWX2,Y2
870 IF HF=1 THEN MOVEX3,Y3:
    DRAWX4,Y4
880 HF=0:YC=750-PT*20:X3=104:
    X4=1176: Y3=YC+200*TAN(RL*2*C):
    Y4=YC-200*TAN(RL*2*C)
890 IF HF4=1 THEN PLOT69,RUX,RUY
900 HF4=0
910 RUX=(RX-640)*2+640
920 RUY=(Y4-Y3)/(X4--X3)*RUX+
    Y3-24
930 IF RUX>100 AND RUX<1180 AND
    RUY>500 AND RUY<900 THEN
    PLOT69,RUX,RUY:HF4=1
940 IF (YC>896 OR YC<504) AND RL=0
    THEN 1010
950 IF Y3>896 THEN X3=640-
    (640-X3)*(896-YC)/(Y3-YC):
    Y3=896:GOTO 970
960 IF Y3<504 THEN X3=640-(640-X3)
    *(504-YC)/(Y3-YC):Y3=504
970 IF Y4>896 THEN X4=640-
    (640-X4)*(896-YC)/(Y4-YC):
    Y4=896:GOTO 990
980 IF Y4<504 THEN X4=640-(640-X4)
    *(504-YC)/(Y4-YC):Y4=504
990 IF X3<104 OR X4>1176 THEN 1010
1000 HF=1:MOVE X3,Y3:DRAW X4,Y4
1010 WB=AD: IF AD>180 THEN
    WB=WB-360
1020 IF RB>180 THEN WB=WB+360-RB
    ELSE WB=WB-RB
1030 IF ABS(WB)>60 AND
    ABS(PY)>1000 THEN 1040
1040 RR=RL:PP=PT
1050 ENDPROC
1060 DEF PROCKEY
1070 TC2=TC
1080 IF RND(10)=1 AND TC>.2 THEN
    TC=TC-.2
1090 IF RND(10)=1 AND TC<8.8 THEN
    TC=TC+.2
1100 IF RND(10)=1 THEN PT=PT+1
1110 IF RND(10)=1 THEN PT=PT-1
1120 IF RND(10)=1 AND RL>-30 THEN
    RL=RL-1
1130 IF RND(10)=1 AND RL<30 THEN
    RL=RL+1
1140 ENDPROC
1150 PP=-1:RR=-1:RL=0:PT=0:
    AD=0:HF=0:HF2=0:HF3=0:
    HF4=0
1160 ST=0:VX=0:VY=0:VZ=0:
    BC=0:NC=0
1170 C=PI/180:PX=0:PY=-20000:
    PZ=2000:AS=150
1190 X0=0:X1=0
1200 X0=X0/3
1210 CLS
1220 WY=-X0*COS(X1*C)
1230 WX=-X0*SIN(X1*C)
1240 GZ=VZ:GY=VY+WY:GX=VX+WX
1250 TC=5
1260 RT=3:TP=5:WR=50:HD=30000
1270 PROCSCREEN
1280 AS2=AS:TC2=TC:TN2=
    PZ/1000:UN2=PZ-1000*INT
    (PZ):PROCINIT
1290 PROCKEY:PROCWORKOUT:PROCINFO
1300 IF PZ<=0 THEN 1320
1310 GOTO 1290
1320 END
```

Line 15 sends you to Line 1150, which, with the next ten lines, sets the variables and the position of the aeroplane in the sky. Line 1170 starts the aeroplane 2000 metres up in the sky, and 20,000 metres away from the

target runway. Line 119Ø sets the wind speed and direction to Ø—next time you will enter new commands that allow you to choose the force and angle of the wind for yourself. Line 124Ø sets the variables that control the position of the aeroplane in the sky according to its speed in any direction, and the strength and direction of the wind. PZ tells how far the aeroplane is along the Z axis—up and down; VZ refers to the up and down velocity; VY is the velocity forwards or backwards (along the Y axis), WY is the windspeed in the same direction. GX, VX and WX refer to the distance moved, velocity and windspeed along the X axis—that is, right to left. Line 125Ø sets the rev counter, and Line 126Ø defines the limits of roll, pitch, and the runway width. Line 127Ø sends you to the procedure that draws the cockpit on the screen.

PROCINIT, which starts on Line 55Ø and ends at Line 58Ø draws the dial pointers on the screen. PROCKEY, in this part, consists of temporary random flight commands.

PROCWORKOUT updates the variables as the aeroplane flies. Lines 42Ø and 43Ø check to see if you have allowed the airspeed to fall below 30 metres per second—in which case the aeroplane stalls, and you are sent to subroutine 52Ø, which ends at Line 54Ø. Line 53Ø introduces a random element.

PROCINFO redraws the dials and counters as they need to be updated. Lines 6ØØ to 67Ø, rub out the hands, and then redraw them in the new position. Lines 68Ø and 69Ø change the value of the runway bearing, and Line 7ØØ prints out all the new numbers on the counters. Lines 71Ø to 77Ø move the radar image of the runway, while Lines 78Ø to 86Ø draw the artificial horizon.

The real horizon is drawn by Lines 87Ø, 88Ø, and 94Ø to 1ØØØ. Lines 89Ø to 93Ø position the dot that indicates the runway.

```
500 RA=AD*C:VX=AS*SIN(RA)
510 VY=AS*COS(RA):RETURN
1000 PZ=PZ+GZ:PY=PY+GY:
   PX=PX+GX
1020 AS=AS+16*(TC*30-AS-
   8*PT)/AS
1030 IFST=1 THENPMODE4,1:
   DRAW"BM108,40C0":A$="STALL":
   GOSUB4000:DRAW"C5":ST=0:
   GOTO1050
1040 IF AS<30 GOSUB1500
1050 AD=AD+RL:IF AD<0 THEN
   AD=AD+360
1060 IF AD>359 THEN AD=AD-360
1070 VZ=AS*SIN(PT*C)-10+AS/15
1080 GZ=VZ:GY=VY+WY:GX=
   VX+WX
1090 IF VY=0 THEN GD=-PI/2:
   GOTO1110
1100 GD=-ATN(VX/VY)/C
1110 GOSUB500
1120 RETURN
1500 PMODE4,1:ST=1:DRAW"BM108,
   40":A$="STALL":GOSUB4000:
   PLAY"T20AGFEDCAGFEDC"
1510 RL=RND(21)-10:PT=-20-
   RND(5)
1520 RETURN
2000PCOPY5TO7:PCOPY6TO8:
   PMODE4,5
2010 LINE(35,120)-(35+20*SIN
   (AS*PI/200),120-20*COS
   (AS*PI/200)),PSET
2020 TN=PZ/1000:UN=PZ-1000*
   INT(TN):LINE(155,120)-
   (155+15*SIN(TN*PI/5),120-
   15*COS(TN*PI/5)),PSET
2030 LINE(155,120)-(155+20*SIN
   (UN*PI/500),120-20*COS
   (UN*PI/500)),PSET
2040 LINE(215,120)-(215+20*SIN
   (TC*PI/5),120-20*COS
   (TC*PI/5)),PSET
2050 DRAW"BM16,172S8":A$=STR$
   (ABS(INT(AD))):GOSUB4000
2060 IF PY=0 THEN RB=0 ELSE
   RB=ATN(PX/PY)/C:IF PY>0 THEN
   RB=RB+180
2070 IF RB<0 THEN RB=RB+360
2080 DRAW"BM80,172":A$=STR$
   (INT(RB)):GOSUB4000:DRAW
   "BM140,172":A$=STR$(ABS
   (INT(PX))):GOSUB4000
2090 DRAW"BM196,172':A$=STR$
   (INT(SQR(PY*PY+PX*PX))):
   GOSUB4000
2100 YC=120+PT:X1=80:X2=110:
   Y1=YC+17*TAN(RL*2*C):
   Y2=YC-17*TAN(RL*2*C)
2110 IF(YC<105ORYC>135)AND
   RL=0THEN2320
2120 IF Y1<105 THEN X1=95-
   (95-X1)*(105-YC)/(Y1-YC):
   Y1=105:GOTO2140
2130 IF Y1>135 THEN X1=95-
   (95-X1)*(135-YC)/(Y1-YC):
   Y1=135
2140 IF Y2<105 THEN X2=95-
   (95-X2)*(105-YC)/(Y2-YC):
   Y2=105:GOTO2160
2150 IF Y2>135 THEN X2=95-
   (95-X2)*(135-YC)/(Y2-YC):
   Y2=135
2160 IFX1 <80ORX2>110 THEN2180
2170 LINE(X1,Y1)-(X2,Y2),PSET
2180 PMODE4,1:IFX5-R>10AND
   X5+R<245ANDY5>0ANDY5<80
   THENCIRCLE(X5,Y5),R,0,10,0,.5
2190 IF RL=RR AND PP=PT THEN2290
2200 IFHF=1 THENLINE(X3,Y3)-
   (X4,Y4),PRESET
2210 HF=0:YC=33+PT*4:X3=11:
   X4=244:Y3=YC+118*TAN
   (RL*2*C):Y4=YC-118*TAN
   (RL*2*C)
2220 IF(YC<100RYC>79)ANDRL=0
   THEN2290
2230 IFY3<1THENX3=128-(128-X3)*
   (1-YC)/(Y3-YC):Y3=1:GOTO2250
2240 IFY3>79THENX3=128-(128-
   X3)*(79-YC)/(Y3-YC):Y3=79
2250 IFY4<1THENX4=128-(128-X4)*
   (1-YC)/(Y4-YC):Y4=1:GOTO2270
2260 IFY4>79THENX4=128-(128-
   X4)*(79-YC)/(Y4-YC):Y4=79
2270 IFX3<11ORX4>244THEN2290
2280 HF=1:LINE(X3,Y3)-(X4,Y4),PSET
```



```
229Ø WB=AD:IFAD>18ØTHENWB=
    WB-36Ø
23ØØ IFRB>18ØTHENWB=WB+36Ø-
    RB ELSEWB=WB-RB
231Ø IFABS(WB)>6Ø ANDABS(PY)
    >1ØØØ THEN237Ø
232Ø AN=118/(6Ø*SQR((X3-X4)*
    (X3-X4)+(Y3-Y4)*(Y3-Y4))):
    X5=(X3+X4)/2+SGN(X3-X4)+
    WB*AN*(X3-X4)
233Ø Y5=(Y3+Y4)/2+2+WB*AN*
    (Y3-Y4):IFX5<11ORX5>244
    ORY5<1ORY5>79THEN237Ø
234Ø IFABS(PY)<1ØØØ THENR=8-
    Y5/1Ø ELSER=4ØØØ/ABS(PY):
    IFR*1Ø+Y5>8Ø THENR=8-Y5/1Ø
235Ø IFY5<1ORY5>79ORX5-R
    <11ORX5+R>244THEN237Ø
236Ø CIRCLE(X5,Y5),R,5,1Ø,Ø,.5
237Ø PCOPY7TO3:PCOPY8TO4
238Ø RR=RL:PP=PT:RETURN
5Ø8Ø GZ=VZ: GY=VY+WY: GX=VX+WX
5Ø9Ø TC=5
51ØØ RT=3:TP=5:WR=5Ø
55ØØ IF RND(1Ø)=1 THEN RL=
    RL-SGN(RL)+(RND(5)-3)
55Ø5 IF RND(1Ø)=1 THEN
    PT=PT+3-RND(2)*2
551Ø GOSUB 1ØØØ: IF PZ<Ø THEN 553Ø
552Ø GOSUB 2ØØØ: GOTO 55ØØ
553Ø GOTO 553Ø
```

The first commands to be followed by the computer in this part of the program, are contained in the Lines 5Ø8Ø to 553Ø. Line 5Ø8Ø sets the variables that control the position of the aeroplane in the sky, according to its speed in any direction, and the strength and direction of the wind. GZ tells how far the aeroplane has moved along the Z (up and down) axis; VZ refers to the up and down velocity; VY is the velocity forwards or backwards (along the Y axis), WY is the windspeed in the same direction; GX, VX and WX refer to the distance moved, velocity and windspeed along the X axis—that is, right to left.

Line 5Ø9Ø sets the rev counter, and Line 51ØØ defines the limits of the roll (RT), the pitch (TP) and the width of the runway (WR).

The last five lines are the temporary commands. These cause the aeroplane to fly rather crazily, as pitch and roll are randomly calculated.

The subroutine from Line 1ØØØ to 112Ø updates all the variables as the aeroplane flies. Lines 1Ø3Ø and 1Ø4Ø check to see if you have allowed your aeroplane to fall below the stalling speed of 30 metres per second, and if so, sends you to the subroutine that runs from Lines 15ØØ to 152Ø, which recreates the effect of a stall. Line 151Ø introduces a random element into the stall—it could merely plummet like a stone to the ground, or go into a spin as it does so. The subroutine 5ØØ to 51Ø, to which you are sent by Line 111Ø, works out the angle the aeroplane is flying at.

The subroutine that begins at Line 2ØØØ, and ends at Line 238Ø, makes up the remainder of this part of the program. This updates the screen by adjusting the instrument panel, which displays the changing position and movement of the aeroplane.

Line 2ØØØ transfers the original version of the cockpit elsewhere in memory. The airspeed dial is updated by Line 2Ø1Ø, while Lines 2Ø2Ø and 2Ø3Ø affect the altitude dial. The rev counter is changed by Line 2Ø4Ø, while Line 2Ø5Ø updates the bearing of the aeroplane. The runway bearing is changed by Lines 2Ø6Ø to 2Ø8Ø, and the next line, 2Ø9Ø, displays the new value for the distance.

Line 219Ø checks to see whether the horizon should be updated, and Lines 21ØØ to 217Ø draw the new artificial horizon in the dial, while Lines 22ØØ to 228Ø draw the real horizon, if it can be seen through the window.

The new position of the runway is calculated by the Lines 229Ø to 236Ø. Line 235Ø checks to see whether the runway can be drawn on the screen yet. The runway increases in size as the aeroplane approaches, and Line 218Ø blots out the previous runway as the bigger one is drawn.

These changes have been made invisibly, one by one, on an unseen graphics page. Line 237Ø transfers all the redrawn dials and counters to the screen using PCOPY, so that they are seen to move and change simultaneously.



WHAT GOES UP MUST COME DOWN

'I shot an arrow into the air, It fell to earth, I knew not where' Here's how to get your computer to find out, plus routines that could build into exciting action games

There is a particular beauty in watching a computer plot the path of an object in flight, and there are many instances—particularly in games programming—where this type of programming is suitable. Without the use of a computer, plotting such paths would be extremely tedious and liable to many errors. In only a few lines of programming, however, you can get your micro to carry out accurately the vast number of calculations necessary to produce startling results.

The ways in which objects move can depend on several factors, and it is important that at least some of these are taken into account if an action program is to give a realistic effect.

One important special case concerns the motion of an object as it rises from the Earth's surface and is slowed by gravity—or as it falls and is speeded up by gravity. This article shows how to program the motion of projectiles—objects that move horizontally at constant speed (ignoring friction in the air) *and* vertically under the influence of gravity.

One reason why fighting games set in the free fall vacuum of space are so common is that in space you can ignore the effects of gravity and air friction on the motion of both space vehicles and shots. That is not to say there is no gravity in space; there is, but usually its effects are so small that it can reasonably be ignored or it can be assumed that all objects are affected by the same gravity field.

By contrast, battles on Earth that involve thrown or shot objects have to take account of gravity and, often, friction due to air and the effects of wind. Besides military games, there are many instances for which a knowledge of projectile motion is essential for realistic programming. Among these are many sports and athletics simulations, including throwing and kicking balls, diving, darts and all sorts of jumping.

All the projectiles in these instances (whether people or things) have one thing in common—they travel in a trajectory or path which belongs to a group of curves known as parabolas. Writing a program to simulate movement in a parabolic path is not difficult, requiring only an understanding of the forces that cause the motion, and the use of some elementary maths.

Knowing that the motion of a thrown ball, for example, results from a combination of motions in two separate directions gives sufficient clues to solve the problem of programming. One of these directions is along the horizontal or X axis. The other is up and down the Y axis. Throughout this article, it is assumed that a projectile moves at constant speed in the X direction. In fact, an actual object would be slowed by friction in the air, but this is a complication best avoided for all but the most accurate work.

HORIZONTAL MOTION

Enter the first program to simulate horizontal motion, but do not confuse 'I' with '1'. All the programs are for the machines' standard BASIC except for the Vic 20 which needs a Super Expander.

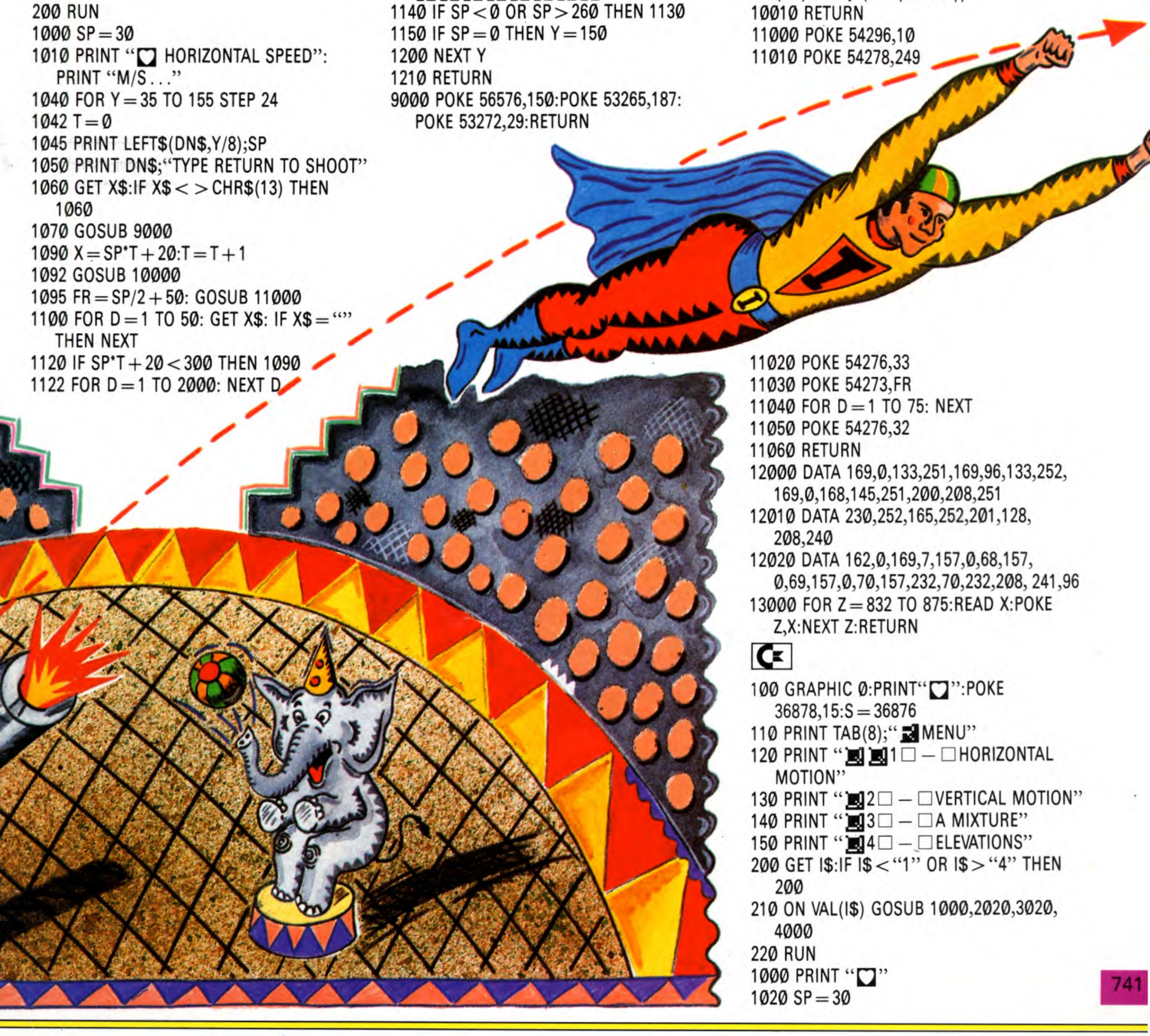
```
100 BORDER 7: PAPER 7: INK Ø: CLS
105 POKE 23658,8
110 PRINT INVERSE 1;AT 2,12;"□MENU□"
120 PRINT AT 6,5;"1: — □PURE
    HORIZONTAL MOTION"
130 PRINT AT 8,5;"2: — □PURE VERTICAL
    MOTION"
140 PRINT AT 1Ø,5;"3: — □A MIXTURE"
150 PRINT AT 12,5;"4: —
    □ELEVATIONS"
200 LET I$=INKEY$: IF I$="" THEN GOTO
    2ØØ
205 IF I$<"1" OR I$>"4" THEN GOTO
    2ØØ
210 GOSUB VAL I$*1ØØØ
220 RUN
1ØØØ CLS
1Ø2Ø LET SP=3Ø
1Ø3Ø PRINT "HORIZONTAL SPEED M/S..."
1Ø4Ø FOR R=124 TO 28 STEP −16
1Ø45 LET T=Ø
1Ø5Ø PRINT AT 21−(R/8),Ø;SP
1Ø6Ø INPUT "PRESS ENTER TO SHOOT",
    LINE Z$
1Ø9Ø LET X=SP*T: LET T=T+1
11ØØ PLOT 3Ø+X,R: BEEP .1,R/4
111Ø PAUSE 1Ø
112Ø IF 3Ø+SP*T<25Ø THEN GOTO 1Ø9Ø
115Ø IF R=28 THEN GOTO 12ØØ
116Ø INPUT "NEW SPEED (Ø END)", LINE I$
1165 LET SP=VAL I$
117Ø IF SP<Ø OR SP>1ØØØ THEN GOTO
    116Ø
119Ø IF SP=Ø THEN LET R=28
12ØØ NEXT R
121Ø RETURN
```

```
1ØØ DN$="▤▣▣▣▣▣▣
    ▣▣▣▣▣▣▣▣▣▣▣▣
    ▣▣":GOSUB 13ØØØ
11Ø PRINT "♥"
12Ø PRINT SPC(15);"◪MENU":PRINT
13Ø PRINT SPC(8);"1□PURE HORIZONTAL
    MOTION▣"
14Ø PRINT SPC(8);"2□PURE VERTICAL
    MOTION▣"
15Ø PRINT SPC(8);"3□A MIXTURE▣"
```



- PLOTTING PROJECTILE PATHS
- EFFECTS OF DIFFERENT GRAVITATIONAL FIELDS
- SIMULATING PARABOLAS
- HOW THE PROGRAMS WORK
- VERTICAL MOTION
- HORIZONTAL MOTION
- COMBINING ROUTINES
- CHANGING THE ANGLE

```
160 PRINT SPC(8);"4□ELEVATIONS■"
170 GET I$:IF I$<"1" OR I$>"4" THEN 170
180 SYS 832
190 ON VAL(I$) GOSUB 1000,2020,3020,
   4040
200 RUN
1000 SP=30
1010 PRINT "■ HORIZONTAL SPEED":
   PRINT "M/S..."
1040 FOR Y=35 TO 155 STEP 24
1042 T=0
1045 PRINT LEFT$(DN$,Y/8);SP
1050 PRINT DN$;"TYPE RETURN TO SHOOT"
1060 GET X$:IF X$<>CHR$(13) THEN
   1060
1070 GOSUB 9000
1090 X=SP*T+20:T=T+1
1092 GOSUB 10000
1095 FR=SP/2+50: GOSUB 11000
1100 FOR D=1 TO 50: GET X$: IF X$=""
   THEN NEXT
1120 IF SP*T+20<300 THEN 1090
1122 FOR D=1 TO 2000: NEXT D
1124 GOSUB 9500
1126 IF Y=155 THEN 1200
1130 PRINT DN$;"□□□NEW SPEED□(0
   □END)":INPUT"□□□□□□□□□
   ■■■■■■■■■■■■■■■■■■■";SP
1140 IF SP<0 OR SP>260 THEN 1130
1150 IF SP=0 THEN Y=150
1200 NEXT Y
1210 RETURN
9000 POKE 56576,150:POKE 53265,187:
   POKE 53272,29:RETURN
9500 POKE 56576,151:POKE 53265,27:
   POKE 53272,21:RETURN
10000 BY=24576+(YAND248)*40+
   (XAND504)+(YAND7):POKEBY,PEEK
   (BY)OR2↑(7-(XAND7))
10010 RETURN
11000 POKE 54296,10
11010 POKE 54278,249
11020 POKE 54276,33
11030 POKE 54273,FR
11040 FOR D=1 TO 75: NEXT
11050 POKE 54276,32
11060 RETURN
12000 DATA 169,0,133,251,169,96,133,252,
   169,0,168,145,251,200,208,251
12010 DATA 230,252,165,252,201,128,
   208,240
12020 DATA 162,0,169,7,157,0,68,157,
   0,69,157,0,70,157,232,70,232,208, 241,96
13000 FOR Z=832 TO 875:READ X:POKE
   Z,X:NEXT Z:RETURN
```

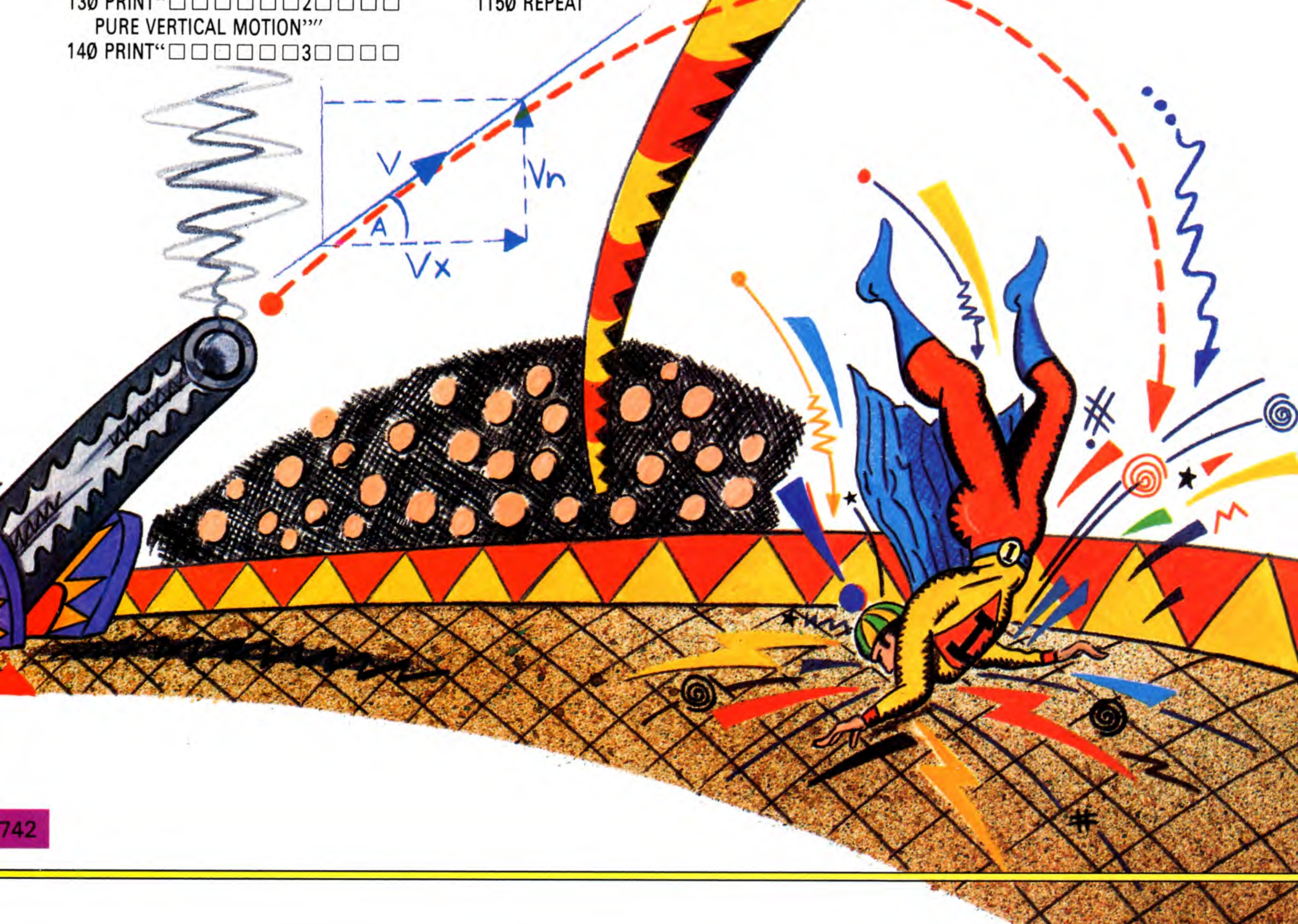


```
100 GRAPHIC 0:PRINT"■":POKE
   36878,15:S=36876
110 PRINT TAB(8);"■MENU"
120 PRINT "■■1□-□HORIZONTAL
   MOTION"
130 PRINT "■2□-□VERTICAL MOTION"
140 PRINT "■3□-□A MIXTURE"
150 PRINT "■4□-□ELEVATIONS"
200 GET I$:IF I$<"1" OR I$>"4" THEN
   200
210 ON VAL(I$) GOSUB 1000,2020,3020,
   4000
220 RUN
1000 PRINT "■"
1020 SP=30
```

```
1Ø4Ø FOR R=1 TO 6:PRINT
    "♡HORIZONTAL SPEED M/S ..."
1Ø5Ø PRINT SP:T=Ø
1Ø6Ø INPUT "■HIT RETURN TO SHOOT";Z$
1Ø7Ø GRAPHIC 2
1Ø9Ø X=SP*T:T=T+1
11ØØ POINT 1,X,5ØØ:POKE 36876,128+
    (X/1Ø)
112Ø IF SP*T<1Ø23 THEN 1Ø9Ø
113Ø POKE S,Ø:FOR Z=1 TO 1ØØØ:NEXT
    Z:GRAPHIC Ø
115Ø IF R=6 THEN 12ØØ
116Ø I$="":INPUT "♡NEW SPEED (Ø
    END)";I$
1165 SP=VAL(I$):IF LEN(I$)=Ø THEN 116Ø
117Ø IF SP<Ø OR SP>1Ø23 THEN 116Ø
119Ø IF SP=Ø THEN RETURN
12ØØ NEXT R
121Ø RETURN
```

```
1ØØ MODE1
11Ø PRINT TAB(15,2);"□□MENU";
    TAB(42);"===="''''
12Ø PRINT"□□□□□□1□□□□
    PURE HORIZONTAL MOTION"''
13Ø PRINT"□□□□□□2□□□□
    PURE VERTICAL MOTION"''
14Ø PRINT"□□□□□□3□□□□
    A MIXTURE"''
15Ø PRINT"□□□□□□4□□□□
    ELEVATIONS"
16Ø REPEAT:I$=GET$:UNTIL I$>"Ø" AND
    I$<"5"
17Ø GOSUB VAL I$*1ØØØ:RUN
1ØØØ MODE5
1Ø1Ø VDU19,3,Ø,Ø,Ø,Ø,19,Ø,2,Ø,Ø,Ø
1Ø2Ø SP=3Ø
1Ø3Ø PRINT' "HORIZONTAL SPEED"'"m/s..."
1Ø4Ø FOR R=8ØØ TO 2ØØ STEP -96
1Ø45 T=Ø
1Ø5Ø PRINT TAB(Ø,7+(8ØØ-R)/32);SP□
    TAB(Ø,3Ø);"RETURN TO SHOOT";
1Ø6Ø B=GET
1Ø7Ø PRINT TAB(Ø,3Ø);SPC(16)
1Ø8Ø REPEAT
1Ø9Ø X=SP*T:T=T+1
11ØØ PLOT69,192+X,R-16:
    SOUND1,-15,R/4,2
111Ø D=INKEY(85)
112Ø UNTIL SP*T+192
    >11ØØ
113Ø IF R<25Ø
    THEN 12ØØ
114Ø D=INKEY(1ØØ)
115Ø REPEAT
116Ø INPUT TAB(Ø,29)"NEW SPEED(Ø
    END):"I$:PRINT TAB(Ø,29);SPC(3Ø);
1165 SP=VAL I$
117Ø UNTIL LEN I$>Ø AND SP>=Ø AND
    SP<1ØØØ
119Ø IF SP=Ø THEN R=Ø
12ØØ NEXT
121Ø RETURN
```

```
1ØØ CLS:PMODE3
11Ø PRINT@13,"menu"
12Ø PRINT@131,"1- PURE HORIZONTAL
    MOTION"
13Ø PRINT@195,"2- PURE VERTICAL
    MOTION"
14Ø PRINT@259,"3- A MIXTURE"
15Ø PRINT@323,"4- ELEVATIONS"
16Ø A$=INKEY$:IF A$<"1" OR A$>"4"
    THEN 16Ø
17Ø ON VAL(A$) GOSUB 1ØØØ,2ØØØ,
    3ØØØ,4ØØØ
18Ø A$=INKEY$
19Ø IF INKEY$="" AND PEEK(65314)
    <>7 THEN 19Ø ELSE RUN
```



```
1000 PCLS
1010 LINE(0,0)-(255,191),PSET,B
1020 SP=30
1030 CLS:PRINT"HORIZONTAL
    SPEED":PRINT"M/S ..."
1040 FOR R=39 TO 159 STEP 24:T=-1
1050 PRINT@32*INT(R/12)-1,SP;
    "*":PRINT@448,"ENTER TO SHOOT"
1060 IF INKEY$<>CHR$(13) THEN 1060
1070 SCREEN1,0
1090 T=T+1:X=SP*T
1100 PSET(30+X/5,R,2):SOUNDR,1
1105 D=50
1110 IF PEEK(345)=255 AND D>0 THEN
    D=D-1:GOTO1110
1120 IF SP*(T+1)/5<223 THEN 1090
1130 IF R>150 THEN 1200
1135 D=200
1139 A$=INKEY$
1140 IF INKEY$="" AND D>0 THEN
    D=D-1:GOTO1140
1160 PRINT@448,"":PRINT@448,"NEW
    SPEED (0□END) :";:INPUT SP
1170 IF SP<0 OR SP>999 THEN 1160
1190 IF SP=0 THEN R=160
1200 NEXT
1210 RETURN
```

RUN the program to see a menu of four options. As yet, you have only keyed the routine for the first option, so if you press 2, 3 or 4, you will get an error message. When you press 1, the program branches to the routine beginning at Line 1000.

This routine uses a FOR ... NEXT loop (Lines 1040 to 1200) to let you throw an object horizontally at six different speeds. The first time round the loop, a speed of 30 metres per second (m/s) is chosen automatically and simulated as a series of points in a straight line.

After the first pass round the loop, you are prompted to enter a different value for speed, but you can escape from the loop by entering 0, when the program RUNs again and displays the menu. Enter a value—for example, 60—and press RETURN or ENTER again to shoot. You can speed up the action by holding down or repeatedly pressing this key (the space bar on Commodores, Dragon and Tandy). Now compare the result with the 30 m/s line. Enter and compare five other speeds, after which the display will return to the menu.

The section of program that PLOTs the points lies between Lines 1090 and 1120. The variable T simulates time which, in this case increases by steps of one second, so the points are spaced out regularly (necessary for constant speed) in the horizontal (X axis) direction. The actual separation of points is set by the term SP*T at Line 1090. This term ensures that the greater the speed (SP), the greater is the separation of the points.

You may recognize SP*T as part of the formula Distance Is Speed times Time, as it is taught in school physics lessons. From this it is clear that the program plots distances (the space between points) to simulate speed.

VERTICAL MOTION

Enter the next few lines, which give you a routine to simulate motion in the vertical direction:

```
2000 CLS
2020 LET G=10: LET SP=50
2030 PRINT '"GRAV. ACCELERATION:
    m/s/s..."
2040 FOR R=3 TO 18 STEP 3
2050 PRINT AT 20,R;"*";AT 21,R;G
2060 INPUT "ENTER TO SHOOT", LINE I$
2080 FOR T=0 TO 250 STEP .5
2090 LET H=SP*T-0.5*G*T*T
2100 IF H>143 THEN BEEP .05,10: PAUSE
    50: NEXT T: GOTO 2110
2105 IF H>=0 THEN PLOT R*8+4,H+32:
    BEEP .05,H/4: PAUSE 40: NEXT T
2110 IF R>15 THEN GOTO 2180
2140 INPUT "NEW G (0 END)", LINE I$
2142 IF LEN I$=0 THEN GOTO 2140
2145 LET G=VAL I$
2150 IF NOT (LEN I$>0 AND G>=0 AND
    G<400) THEN GOTO 2140
2170 IF G=0 THEN LET R=18
2180 NEXT R
2190 RETURN
```

```
2020 G=10:SP=50
2025 PRINT"♥GRAV ACCELERATION: M/S/S
    ..."
2040 FOR X=40 TO 280 STEP 48
2042 PRINT DN$;SPC(X/8);"*▟■■■■";G
2045 PRINT "RETURN TO SHOOT"
2050 GET I$: IF I$<>CHR$(13) THEN 2050
2060 GOSUB 9000
2070 FOR T=0 TO 250 STEP 0.25
2090 H=SP*T-0.5*G*T*T
2095 Y=179-H
2100 IF H>0 AND H<180 THEN GOSUB
    10000: FR=H/2+50: GOSUB 11000
2102 IF H>=180 THEN FR=250: GOSUB
    11000
2105 FOR D=1 TO 50: GET X$: IF X$=""
    THEN NEXT
2108 IF H<0 THEN T=250
2110 NEXT T
2120 FOR D=1 TO 2000: NEXT
2125 GOSUB 9500
2127 IF X=280 THEN 2170
2130 PRINT DN$:PRINT
2140 PRINT "□□NEW G□(0□END)":
    INPUT"□□□□□□□□■■■■■■
    ■■■■■■■■■■■";G
2150 IF G<0 OR G>200 THEN 2130
2160 IF G=0 THEN X=280
2170 NEXT
2180 RETURN
```

```
2020 G=10:SP=50
2040 FOR R=1 TO 6 KEEP: GRAPHIC 0
2045 PRINT "♥GRAV.ACCELERATION:
    □□□□M/S/S▟"
2050 PRINT G
2060 INPUT "▟RETURN TO SHOOT";
    Z$:GRAPHIC 2
2080 FOR T=0 TO 250 STEP.5
2090 H=SP*T-.5*G*T*T
2100 IF H>1023 THEN POKE S,250:NEXT
    T:GOTO 2110
2105 IF H>=0 THEN:POINT 1,512,
    1023-H:POKE 36876,128+(H/10):
    NEXT T
2110 IF R=6 THEN POKE S,0:GOTO 2180
2140 POKE S,0:FOR Z=1 TO 1000:NEXT
    Z:GRAPHIC 0:I$="":INPUT "♥NEW G
    (0 END)";I$
2142 IF LEN(I$)=0 THEN 2140
2145 G=VAL(I$)
2150 IF LEN(I$)=0 OR G<0 OR G>400
    THEN 2140
2170 IF G=0 THEN R=6
2180 NEXT R
2190 RETURN
```

```
2000 MODE5
2010 VDU19,3,0,0,0,0,19,0,6,0,0,0,24,0;
    128;1280;975;
2020 G=10:SP=50
2030 PRINT' "GRAV. ACCELERATION:
    m/s/s..."
2040 FOR R=3 TO 18 STEP 3
2050 PRINT TAB(R,28);"*"
    TAB(R-1,29);G
2060 PRINT TAB(0,30)"RETURN TO SHOOT":
    REPEAT:D=GET:UNTIL D=13
2070 PRINT TAB(0,30);SPC(16)
2080 FOR T=0 TO 250 STEP.5
2090 H=SP*T-0.5*G*T*T
2100 IF H>220 THEN SOUND1,-15,
    250,2:D=INKEY(50):NEXT T ELSE
    IF H>=0 THEN PLOT69,(R+0.5)
    *64,H*4+128:SOUND1,-15,H/4,2:
    D=INKEY(85):NEXT T
2110 IF R>15 THEN 2180
2120 D=INKEY(100)
2130 REPEAT
2140 INPUT TAB(0,30)"NEW G (0 END):" I$:
    PRINT TAB(0,30);SPC(19)
2145 G=VAL I$
```

```
215Ø UNTIL LEN I$>Ø AND G>=Ø AND
    G<4ØØ
217Ø IF G=Ø THEN R=18
218Ø NEXT R
219Ø RETURN
```

```
2ØØØ PCLS
2Ø1Ø LINE(Ø,Ø)-(255,191),PSET,B
2Ø2Ø G=1Ø:SP=5Ø:CLS
2Ø3Ø PRINT@32,"GRAV.
    ACCELERATION:":PRINT"M/S/S..."
2Ø4Ø FOR R=Ø TO 25 STEP 5
2Ø5Ø PRINT@416+R,"*"+MID$
    (STR$(G),2);
2Ø6Ø PRINT@448,"ENTER TO SHOOT"
2Ø65 IF INKEY$<>CHR$(13) THEN 2Ø65
2Ø7Ø SCREEN1,Ø
2Ø8Ø FOR T=Ø TO 1ØØ STEP .5
2Ø9Ø H=SP*T-.5*G*T*T
2Ø95 IF H<Ø THEN T=25Ø:GOTO 21Ø8
21ØØ IF H>159 THEN SOUND25Ø,1:
    D=3Ø ELSE PSET(1Ø+R*8,16Ø-
    H,2):SOUNDH+1Ø,1:D=5Ø
21Ø5 IF PEEK(345)=255 AND D>Ø THEN
    D=D-1:GOTO 21Ø5
21Ø8 NEXT
211Ø IF R>2Ø THEN 218Ø
2115 D=8Ø
2119 A$=INKEY$
212Ø IF INKEY$="" AND D>Ø THEN
    D=D-1:GOTO212Ø
214Ø PRINT@448,"":PRINT@448,
    "NEW G (Ø END):";:INPUT G
215Ø IF G<Ø OR G>4ØØ THEN 214Ø
217Ø IF G=Ø THEN R=26
218Ø NEXT
219Ø RETURN
```

RUN the program and this time enter 2 to select the second option on the menu. Now press ENTER or RETURN to see a series of points plotted vertically up the screen. Notice, however, that the points are not equally spaced, but instead get closer towards the top.

This is because the object is slowed down by gravity. And this time, the sound helps to explain what is happening. As the object ascends, the pitch of the sound rises, and as it falls the pitch gets lower.

The routine gives you six trials as before (set at Line 2Ø4Ø) to enter different values for the effect of gravity. This is the variable G, which initially is set to 1Ø (Line 2Ø2Ø) and used at Line 2Ø9Ø.

This relationship is the formula for the distance an object falls under gravity. Its usual form is: $s=ut+\frac{1}{2}gt^2$, where s is the distance (H in this case), t the time and g the *acceleration due to gravity* (G). Notice that T*T is used at Line 2Ø9Ø instead of T↑2, because microcomputers are faster at multiplying than at squaring numbers.

The acceleration of an object is a measure of its change of speed with time. Near the Earth's surface, g has a value approximately equal to 10 m/s/s. This means that the speed of a falling object increases by 10 m/s each second. The speed of an object in ascent decreases by 10 m/s each second. This is a negative acceleration. That is why there is a minus sign (instead of a plus as in the standard formula) at Line 2Ø9Ø.

The expression of g is commonly used in relation to space travel. For example, acceleration of a manned spacecraft taking off for orbit reaches about 10 g. This means that the craft is increasing speed by 10*10 m/s/s or 100 m/s/s. A later article will cover the topic of orbits in more detail.

The first time round the loop starting at Line 2Ø4Ø, the positions of the object are plotted at one-second intervals. The initial speed is 50 m/s and the value for G is 10 m/s/s (both set at Line 2Ø2Ø). As in the first routine, you can speed up the action by holding down or repeatedly pressing ENTER or RETURN (or the space bar). You can then change the value of G, when prompted on the screen, and compare the effect of six different values. Before the loop is complete, you can escape from it by entering Ø, when the program returns to the menu.

COMBINED MOTION

To simulate the motion of a projectile, you need only combine these routines so that the object moves in both horizontal and vertical directions at the same time. Enter the next few lines to set up the third routine:

```
3ØØØ CLS
3Ø2Ø LET G=1Ø: LET SP=5Ø
3Ø3Ø FOR R=1 TO 6
3Ø4Ø PRINT AT Ø,Ø;"G=";G;"m/s/s□"'
    "SPEED=";SP;"m/s□"
3Ø5Ø INPUT "ENTER TO SHOOT", LINE I$
3Ø7Ø FOR T=Ø TO 25Ø STEP .5
3Ø8Ø LET H=SP*SIN ((PI/18Ø)*45)
    *T-.5*G*T*T
3Ø9Ø LET X=SP*COS ((PI/18Ø)*45)*T
31ØØ IF H<175 AND X<255 THEN GOTO
    31Ø5
31Ø2 IF H>Ø THEN BEEP .Ø5,1Ø: PAUSE 25:
    NEXT T: GOTO 311Ø
31Ø3 LET T=25Ø: NEXT T: GOTO 311Ø
31Ø5 IF H>=Ø THEN PLOT X,H:BEEP
    .Ø5,H/4: PAUSE 4Ø: NEXT T: GOTO 311Ø
31Ø6 LET T=25Ø: NEXT T
311Ø IF R=6 THEN GOTO 323Ø
312Ø PAUSE 5Ø
314Ø INPUT "NEW G (Ø END)", LINE I$
315Ø IF LEN I$=Ø THEN GOTO 314Ø
3155 LET G=VAL I$
316Ø IF NOT (LEN I$>Ø AND G>=Ø AND
    G<1ØØØ) THEN GOTO 314Ø
317Ø IF G=Ø THEN LET R=6: GOTO 323Ø
319Ø INPUT "NEW SPEED", LINE I$
3195 LET SP=VAL I$
32ØØ IF NOT (LEN I$>Ø AND SP>Ø AND
    SP<1ØØØ) THEN GOTO 319Ø
323Ø NEXT R
324Ø RETURN
```

```
3Ø2Ø G=1Ø: SP=5Ø
3Ø3Ø FOR R=1 TO 6
3Ø32 PRINT "▢G="G"M/S/S□□□
    SPEED=";SP;" M/S"
3Ø35 PRINT DN$;"TYPE RETURN TO SHOOT"
3Ø4Ø GET I$: IF I$<>CHR$(13) THEN
    3Ø4Ø
3Ø45 GOSUB 9ØØØ
3Ø5Ø FOR T=Ø TO 25Ø STEP Ø.25
3Ø8Ø H=SP*SIN(45*π/18Ø)*T-Ø.5*
    G*T*T
3Ø9Ø X=SP*COS(45*π/18Ø)*T+2Ø
3Ø91 IF H>=Ø AND H<179 AND X<3ØØ
    THEN Y=179-H:GOSUB1ØØØØ:
    FR=H/2+5Ø:GOSUB11ØØØ
3Ø92 IF H>=179 THEN FR=25Ø: GOSUB
    11ØØØ
3Ø93 IF H<Ø THEN T=25Ø
3Ø95 FOR D=1 TO 5Ø: GET X$: IF X$=""
    THEN NEXT
31ØØ NEXT T
31Ø3 FOR D=1 TO 2ØØØ:NEXT
31Ø5 GOSUB 95ØØ
311Ø IF R=6 THEN 323Ø
312Ø PRINT "▣▣▣": INPUT "NEW G□
    (Ø□END)□□□□□□□□□▮▮▮▮
    ▮▮▮▮▮▮▮▮▮▮▮";G
313Ø IF G<Ø OR >1ØØØ THEN 312Ø
314Ø IFG=Ø THEN R=6: GOTO 323Ø
315Ø PRINT "▣▣▣▣▣": INPUT "NEW
    SPEED□";SP
316Ø IF SP<Ø OR SP>1ØØØ THEN
    315Ø
323Ø NEXT R
324Ø RETURN
```

```
3Ø2Ø G=1Ø:SP=5Ø
3Ø3Ø FOR R=1 TO 6:GRAPHIC Ø
3Ø4Ø PRINT "▢G="G;"M/S/S":PRINT
    "▣SPEED=";SP;"M/S"
3Ø5Ø INPUT "▣RETURN TO SHOOT";Z$:
    GRAPHIC 2
3Ø7Ø FOR T=Ø TO 25Ø STEP.5
3Ø8Ø H=SP*SIN((π/18Ø)*45)*T-.5
    *G*T*T
3Ø9Ø X=SP*COS((π/18Ø)*45)*T
```

```
3100 IF H<1023 AND X<1023 THEN 3105
3102 IF H>Ø THEN POKE S,25Ø:NEXT
   T:GOTO 311Ø
3103 T=25Ø:NEXT T:GOTO 311Ø
3105 IFH>=ØTHEN:POINT1,X,1023-H:
   POKES,128+(X/1Ø):NEXTT:GOTO 311Ø
3106 T=25Ø:NEXT T
3110 IF R=6 THEN POKE S,Ø:GOTO 323Ø
3120 POKE S,Ø:FOR Z=1 TO 1ØØØ:NEXT
   Z:GRAPHIC Ø
3140 I$="":INPUT "♥NEW G(Ø END)";I$
3150 IF LEN(I$)=Ø THEN 314Ø
3155 G=VAL(I$)
3160 IF LEN(I$)=Ø OR G<Ø OR G>1ØØØ
   THEN 314Ø
3170 IF G=Ø THEN R=6:GOTO 323Ø
3190 I$="":INPUT "♥NEW SPEED";I$
3195 SP=VAL(I$)
3200 IF LEN(I$)=Ø OR SP<1 OR
   SP>1ØØØ THEN 319Ø
3230 NEXT R
3240 RETURN
```

```
3ØØØ MODE5
3Ø1Ø VDU19,3,3,Ø,Ø,Ø,19,Ø,4,Ø,Ø,Ø,24,Ø;
   128;128Ø;9ØØ;
3Ø2Ø G=1Ø:SP=5Ø
3Ø3Ø FOR R=1 TO 6
3Ø4Ø PRINT TAB(Ø,1);"G=";G;
   "□m/s/s""SPEED=";SP;"□m/s"
3Ø5Ø PRINT TAB(Ø,3Ø)"RETURN TO
   SHOOT□":D=GET
3Ø6Ø PRINT TAB(Ø,3Ø);SPC(16)
3Ø7Ø FOR T=Ø TO 25Ø STEP.5
3Ø8Ø H=SP*SIN RAD 45*T-
   Ø.5*G*T*T
3Ø9Ø X=SP*COS RAD 45*T
31ØØ IF H<22Ø AND X<32Ø THEN 31Ø5
31Ø2 IF H>Ø THEN SOUND1,-15,
   25Ø,2:D=INKEY(5Ø):NEXT:
   GOTO 311Ø
31Ø3 T=25Ø:NEXT:GOTO 311Ø
31Ø5 IF H>=Ø THEN PLOT69,X*4,
   H*4+128:SOUND1,-15,H/4,2:
   D=INKEY(85):NEXT ELSE T=25Ø: NEXT
311Ø IF R=6 THEN 323Ø
312Ø D=INKEY(1ØØ)
313Ø REPEAT
314Ø INPUT TAB(Ø,29)"NEW G (Ø END)□"I$
315Ø PRINT TAB(Ø,29);SPC(19);
3155 G=VAL I$
316Ø UNTIL LEN I$>Ø AND G>=Ø AND
   G<1ØØØ
317Ø IF G=Ø THEN R=6:GOTO 323Ø
318Ø REPEAT
319Ø INPUT TAB(Ø,29)"NEW SPEED:□"I$:
   PRINT TAB(Ø,29);SPC(3Ø);
3195 SP=VAL I$
32ØØ UNTIL LEN I$>Ø AND SP>Ø AND
   SP<1ØØØ
```



Trajectories are changed by altering gravity or the object's speed



Without gravity and air friction, a projectile's velocity is constant

```
323Ø NEXT R
324Ø RETURN
```

```
3ØØØ PCLS
3Ø1Ø LINE(Ø,Ø)-(255,191),PSET,B
3Ø2Ø G=1Ø:SP=5Ø
3Ø3Ø FOR R=1 TO 6:CLS
3Ø4Ø PRINT"G□□□□=";G;"M/S/S":
   PRINT"SPEED=";SP;"M/S"
3Ø5Ø PRINT@448,"ENTER TO SHOOT"
3Ø6Ø IF INKEY$<>CHR$(13) THEN 3Ø6Ø
3Ø65 SCREEN1,Ø
3Ø7Ø FOR T=Ø TO 2ØØ STEP .5
3Ø8Ø H=SP*SIN(ATN(1))*T-.5*G*T*T
3Ø9Ø X=SP*COS(ATN(1))*T
3Ø95 IF H<Ø THEN T=25Ø:GOTO 31Ø6
31ØØ IF X>251 THEN T=25Ø:GOTO
   31Ø6 ELSE IF H>189 THEN SOUND
   25Ø,1:D=25 ELSE PSET(X+2,
   19Ø-H,(R+3)/2):SOUNDH+1Ø,1:D=35
31Ø4 IF PEEK(345)=255 AND D>Ø THEN
   D=D-1:GOTO 31Ø4
31Ø6 NEXT
311Ø IF R>5 THEN 323Ø
3115 D=1ØØ
3119 A$=INKEY$
312Ø IF INKEY$="" AND D>Ø THEN
   D=D-1:GOTO312Ø
313Ø PRINT@416,"":PRINT@448,""
314Ø PRINT@416,"NEW G (Ø END)";:
   INPUT G
316Ø IF G<Ø OR G>9999 THEN 313Ø
317Ø IF G=Ø THEN R=6:GOTO 323Ø
318Ø PRINT@448,""
319Ø PRINT@448,"NEW SPEED:";:
   INPUT SP
32ØØ IF SP<Ø OR SP>999 THEN 318Ø
323Ø NEXT
324Ø RETURN
```

RUN the program and enter 3 in response to the prompt to select the third option. When you press ENTER or RETURN, points are plotted in a curve starting at the bottom left of the screen and ending some way along towards the bottom right. This is the trajectory of an object shot at a speed of 50 m/s in a gravity of 10 m/s/s.

The structure of the routine is similar to that of the previous two. The calculating and plotting sections are in a FOR ... NEXT loop (Lines 3Ø3Ø to 323Ø), which lets you compare six different trajectories, five of which you specify. As in the other trials, you can escape from the routine by entering Ø at this or any subsequent stage, to return to the menu. It is more likely, however, that you wish to enter a new set of values to compare trajectories.

Enter a value of 5 for G and keep SP at 5Ø, then press ENTER or RETURN to shoot. This time the object will go higher and farther. Now keep G at 5, but reduce SP to 25 and compare the result. Continue experimenting, changing both G and SP, and listen to the sounds to help you understand the motion of the object when it disappears from the screen. A note which is increasing in pitch indicates that the object is rising, whereas a decreasing one indicates it is falling.

HOW IT WORKS

Remember that the trajectory of the object is plotted as H coordinates in the Y axis direction, and as X coordinates in the X axis direction. These two coordinates are calculated at Lines 3Ø8Ø and 3Ø9Ø. The only difference is that here the H coordinate has speed (SP) multiplied by the SINE of an angle, and the X coordinate has SP multiplied by the COSine of the same angle (45°). This explains why, when G is small and SP is large, the trajectory is apparently a diagonal line at 45°.

The reason for these trigonometric ratios is to calculate what fraction of the object's motion applies in each of the two directions—vertically and horizontally. These fractions are called components. If the starting speed of the object is 50 m/s, for example, both the vertical and horizontal components are somewhat less than 50. Added together, however, they give exactly 50 m/s.

You will not go wrong if you remember

that the vertical component is SP*SIN A and the horizontal component is SP*COS A. A is the angle of elevation of the gun, bow or whatever.

To understand how these values are arrived at, however, it helps to look at a sketch. The sketch on page 542 shows a projectile starting its motion with actual speed V at an angle A to the horizontal. The dashed lines show the components of speed (Vh and Vx) in the two directions. The sine of angle A is Vh/V, so this relationship can be arranged as Vh = V*SIN A. Similarly, the cosine of A is Vx/V, which when rearranged becomes Vx = V*COS A.

Following this pattern in the program, you need SP*SIN 45 for the vertical component of speed, and SP*COS 45 for the horizontal component.

Except on the Commodores and Spectrum, there is a RAD before 45 in Lines 3Ø8Ø and 3Ø9Ø. This is to convert degrees into radians, the way your computer measures angles. The Commodores achieve the conversion by multiplying the 45 by a factor.

CHANGING ANGLE

At this stage, you may be wondering why the angle should be fixed at 45°, because it limits the range of the projectile. Both the angle of elevation and the initial speed can be varied to change the range of the projectiles. And it is more usual to change the angle only to vary the range, leaving the speed constant. The next routine achieves just this:

```
4ØØØ CLS
4Ø1Ø LET FL=Ø
4Ø2Ø RESTORE : FOR N=Ø TO 7: READ A:
   POKE USR "A"+N,A: NEXT N
4Ø4Ø LET A=7Ø: LET SP=5Ø
4Ø6Ø PRINT AT Ø,Ø;"ANGLE=";A;CHR$
   144;CHR$ 32
4Ø7Ø INPUT "ENTER TO SHOOT", LINE I$
4Ø8Ø FOR T=Ø TO 25Ø STEP .5
4Ø9Ø LET H=SP*SIN
   ((PI/18Ø)*A)*T-.5*1Ø*T*T: LET
   X=5Ø*COS ((PI/18Ø)*A)*T
41ØØ IF H>=Ø THEN PLOT X,H+16: BEEP
   .Ø5,H/4: PAUSE 4Ø: NEXT T: GOTO 411Ø
41Ø5 LET T=25Ø: NEXT T
411Ø PAUSE 5Ø
413Ø INPUT "NEW ANGLE (Ø END)", LINE I$
4135 IF LEN I$=Ø THEN GOTO 413Ø
414Ø LET A=VAL I$
415Ø IF NOT (LEN I$>Ø AND A>=Ø AND
   A<9Ø) THEN GOTO 413Ø
416Ø IF A=Ø THEN LET FL=1
417Ø IF NOT FL THEN GOTO 4Ø6Ø
418Ø RETURN
5ØØØ DATA 24,36,36,36,24,Ø,Ø,Ø
```

```
4Ø4Ø A=7Ø: SP=5Ø
4Ø41 PRINT "■ANGLE=□";A
4Ø42 PRINT DN$;"TYPE RETURN TO SHOOT"
4Ø43 GET I$: IF I$<>CHR$(13) THEN 4Ø43
4Ø44 GOSUB 9ØØØ
4Ø45 FOR T=Ø TO 25Ø STEP Ø.25
4Ø5Ø H=SP*SIN(A*π/18Ø)*T-Ø.5*1Ø*T*T
4Ø6Ø X=SP*COS(A*π/18Ø)*T+2Ø
4Ø7Ø IF H>=Ø THEN Y=179-H: GOSUB
   1ØØØØ: FR=H/2+5Ø: GOSUB 11ØØØ
4Ø75 IFH<Ø THEN T=25Ø
4Ø8Ø FOR D=1 TO 5Ø: GET X$: IF X$=""
   THEN NEXT
4Ø9Ø NEXT T
41ØØ GOSUB 95ØØ
411Ø PRINT "■■■": INPUT "NEW
   ANGLE□(Ø□END)□□□□□□□
   □□■■■■■■■■■■■■■■■■";A
412Ø IF A<Ø OR A>=9Ø THEN 411Ø
413Ø IF A>Ø THEN 4Ø41
414Ø RETURN
```

```
4ØØØ PRINT "■"
4Ø1Ø FL=Ø
4Ø4Ø A=7Ø:SP=5Ø
4Ø6Ø GRAPHIC Ø:PRINT "■ANGLE=";A
4Ø7Ø INPUT "■RETURN TO SHOOT";Z$:
   GRAPHIC 2
4Ø8Ø FOR T=Ø TO 25Ø STEP.5
4Ø9Ø H=SP*SIN((π/18Ø)*A)*T-.5*1Ø*
   T*T:X=5Ø*COS((π/18Ø)*A)*T
41ØØ IF H>=Ø THEN:POINT 1,X*4,1Ø23-
   (H*6):POKE S,128+(H/1Ø):NEXT T:GOTO
   411Ø
41Ø5 T=25Ø:NEXT T
411Ø POKE S,Ø:FOR Z=1 TO 1ØØØ: NEXT Z:
   GRAPHIC Ø
413Ø I$="":INPUT "■NEW ANGLE (Ø
   END)";
   I$
4135 IF LEN(I$)=Ø THEN 413Ø
414Ø A=VAL(I$)
415Ø IF LEN(I$)=Ø OR A<Ø OR A>9Ø
   THEN 413Ø
416Ø IF A=Ø THEN FL=1
417Ø IF NOT FL THEN 4Ø6Ø
418Ø RETURN
```

```
4ØØØ MODE5
4Ø1Ø FL=Ø
4Ø2Ø VDU 23,225,24,36,36,36,24,Ø,Ø,Ø
4Ø3Ø VDU 19,3,4,Ø,Ø,Ø,19,Ø,3,Ø,Ø,Ø,24,
   Ø;128;128Ø;95Ø;
4Ø4Ø A=7Ø: SP=5Ø
4Ø5Ø REPEAT
4Ø6Ø PRINTTAB(Ø,Ø)"ANGLE =□";
   A;CHR$(225)"□"
4Ø7Ø PRINTTAB(Ø,29)"RETURN TO
   SHOOT":D=GET:PRINTTAB(Ø,29)
   SPC(16)
4Ø8Ø FOR T=Ø TO 25Ø STEP.5
4Ø9Ø H=SP*SIN RAD A*T-.5*1Ø*T*T:
   X=SP*COS RAD A*T
41ØØ IF H>=Ø THEN PLOT69,X*5,
   H*4+1ØØ:SOUND1,-15,H/4,2:
   D=INKEY(85):NEXT ELSE T=25Ø:
   NEXT
411Ø D=INKEY(1ØØ)
412Ø REPEAT
413Ø INPUT TAB(Ø,29)"NEW ANGLE (Ø
   END): "I$: PRINTTAB(Ø,29) SPC(26);
414Ø A=VAL I$
415Ø UNTIL LEN I$>Ø AND A>=Ø AND
   A<9Ø
416Ø IF A=Ø THEN FL=1
417Ø UNTIL FL
418Ø RETURN
```



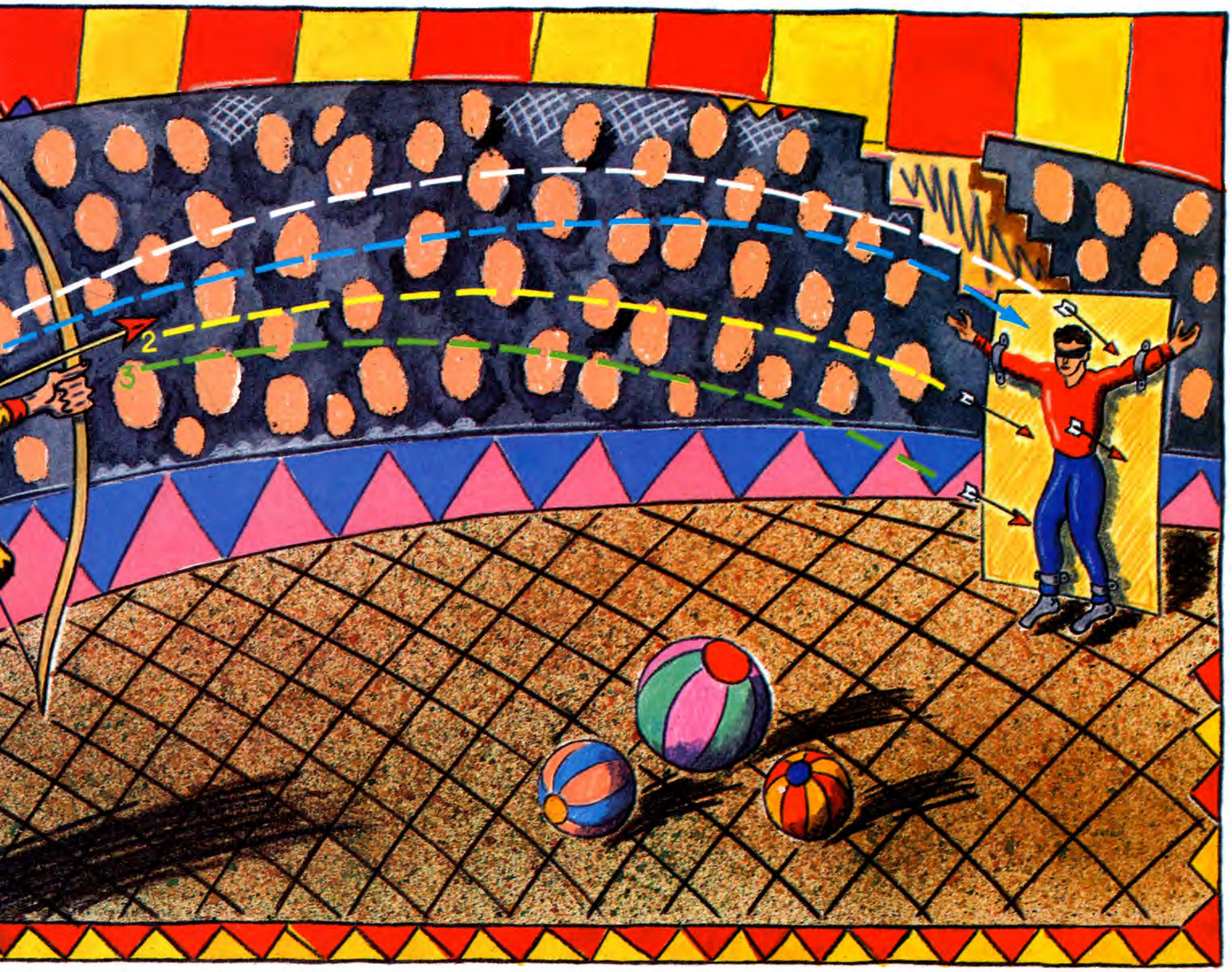
```
4ØØØ PCLS
4Ø2Ø LINE(Ø,Ø)—(255,191),PSET,B
4Ø4Ø A=7Ø: SP=5Ø
4Ø6Ø CLS:PRINT"ANGLE =";A;
    "DEGREES"
4Ø7Ø PRINT@448,"ENTER TO SHOOT"
4Ø72 IF INKEY$<>CHR$(13) THEN 4Ø72
4Ø75 SCREEN1,Ø:AN=A*ATN(1)/45
4Ø8Ø FOR T=Ø TO 25Ø STEP .5
4Ø9Ø H=SP*SIN(AN)*T—.5*1Ø*T*T:
    X=SP*COS(AN)*T
4Ø92 IF H<Ø THEN T=25Ø:GOTO 41ØØ
4Ø94 IF X>251 THEN T=25Ø:GOTO
    41ØØ ELSE IF H>189 THEN SOUND
    25Ø,1:D=25 ELSE PSET(X+2,
    19Ø—H,2):SOUNDH+1Ø,1:D=35
4Ø96 IF PEEK(345)=255 AND D>Ø THEN
    D=D—1:GOTO 4Ø96
41ØØ NEXT
411Ø A$=INKEY$
412Ø IF INKEY$="" THEN 412Ø
413Ø PRINT@448,"NEW ANGLE (Ø END):";:
    INPUT A
416Ø IF A<Ø OR A>=9Ø THEN 412Ø
417Ø IF A>Ø THEN 4Ø6Ø
418Ø RETURN
```

When you RUN the fourth option, you should see the trajectory of an object shot with a speed of 50 m/s and at an elevation of 7Ø° (both set at Line 4Ø4Ø, and used in the relationships at Line 4Ø9Ø—4Ø5Ø and 4Ø6Ø on the Commodore 64).

The routine works as the previous one, except that you can enter as many angles (each between 1° and 89°) as you wish, without returning to the menu. So every time you run through the routine, the variable A at Line 4Ø9Ø is set to the angle you enter. This time the routine is in an infinite loop, so the display will not return to the menu unless you enter Ø as an angle.

Using this routine, try to find the angle that gives the longest range—the angle that lets the object travel farthest in the horizontal direction. You should have no trouble verifying that this angle is 45 degrees.

Most people know this either from experience or intuitively, but it is less well-known that in practice resistance makes a significant difference in most cases, and that 45 degrees gives the longest range only if the starting and landing points are at the same height. Nevertheless, any projectile game that depends only on the player getting the greatest range is doomed to failure, because there will not be a sufficient variation from the ideal angle—which most players will either know or guess. In a subsequent article, you will see how these routines are used as the basis for a challenging and enthralling game.



COMMODORE HI-RES GRAPHICS

Here is a really useful machine code program with a visible result. It extends the standard Commodore BASIC into normally inaccessible high-resolution graphics

So far, many of the painting and drawing programs for the Commodore given in *INPUT* have used Simons' BASIC. This is because you cannot access the machine's high-resolution graphics directly from standard Commodore BASIC. And to use a Simons' BASIC program, you need a special cartridge. But it is possible to write a machine code routine that will give you graphic instructions similar to those used in Simons' BASIC; the following program does just that.

This is the first of several articles which will allow you to RUN all the graphics programs published in *INPUT*, without needing a Simons' cartridge. All you need to do is ensure that the command words are prefixed with an @. It fits into the protected area from CØØØ to CFFF so you don't have to POKE the system variables to shift RAMTOP.

Because it is a long program—too long for *INPUT*'s assembler to cope with in one go—it has been divided up into small chunks, each with their own origin. You'll find it easier to assemble that way.

SETTING UP

The first routine intercepts the computer and makes sure that it goes to this machine code program before doing anything else.

```
     ORG 49152
     LDX #&02
BACK LDA &C00B,X
     STA &73,X
     DEX
     BPL BACK
     RTS
     JMP &C00F
     BYT &00
```

This small routine redirects the Commodore's own CHRGET routine, which starts at &0073, to the beginning of the main routine here. CHRGET is used by BASIC to get each character or token. So you are simply sending it off to perform the program given here, before it deals with any of the regular BASIC commands.

You'll note here that JMP &CØØF is not actually a command in this part of the program. It is data. The rest of the routine picks this instruction byte by byte and stores it in &73, &74 and &75. When that's done, the computer is returned to BASIC. But now, when the computer tries to execute a program, it will be directed to your graphics program first.

BYT ØØ sets aside an empty byte for the temporary storage of error codes.

LOCATING GRAPHIC COMMANDS

To be able to use your graphics routines with this program you have to insert an @ symbol in front of each graphics command. This routine searches through the BASIC program looking for an @ so it can locate the commands.



- ADDING NEW COMMANDS
- WHICH ROUTINE?
- @HIRES
- @COLOUR





```
     ORG 49167
     INC &7A
     BNE ON
     INC &7B
  ON STX &C00E
     TSX
     SEC
     LDA &9D
     BEQ RUN
     LDX #&0D
     JMP &0079
 RUN JSR &0079
     CMP #&40
     BEQ &C030
     LDX &C00E
     JMP &0079
```

The first four instructions move the current BASIC byte pointer onto the next byte of BASIC. This is normally done in the CHRGET routine, but was overwritten when you re-directed it.

LDA &9D and BEQ RUN check to see if a BASIC program is RUNning or not. The system variable at &009D is 0 if one is.

If no program is RUNning, X is loaded with 0D, the code for a syntax error. And JMP &0079 takes it back into the CHRGOT routine which goes on to PRINT the error message on the screen. This routine will not work in direct mode, only with a program.

If a BASIC program is RUNning, the processor branches forward to JSR &0079. Again, this jumps to the CHRGOT subroutine which gets the same byte of BASIC again. But because it has been called as a subroutine, as soon as it has finished, the processor returns to your program, rather than move onto the screen print ROM routine.

The BASIC byte can now be compared to the program token for an @, &40. If it finds one, the processor branches on to the next routine. If not, the current error status is loaded into the X register from its store in &C00E and the processor jumps back to the CHRGOT routine to PRINT the error message or start on the next byte.

STORING POINTERS

This section takes care of the pointers.

```
ORG 49200
JSR &0073
LDX &7A
STX &FB
LDX &7B
STX &FC
LDX #&01
STX &FE
```

The next byte of the BASIC program is obtained from the CHRGET routine. The current byte pointer is stored in &FB and &FC in the user area of the zero page. This is done so that they can be manipulated without corrupting the system variables.

The number 1 is stored in &FE. This location is being used as a counter for the graphic keyword being dealt with.

RECOGNIZING NEW KEYWORDS

Once the @ symbol has been located, the next routine looks at the first letter of the graphics command.

```
      ORG 49215
      LDX #&ØØ
      CMP &CFØØ,X
      BEQ &CØ61
AGAIN LDA #&ØD
      CMP &CFØØ,X
      BEQ STOP
      LDA #&Ø1
      CMP &CFØØ,X
      BEQ &CØA4
      INX
      JMP AGAIN
STOP  INX
      INC &FE
      JSR &ØØ79
      JMP &CØ41
```

A word-search counter X is initialized. The letter in A is compared with those in the table at &CFØØ. When it finds one the same, the processor branches forward to the beginning of the next routine at &CØ61.

If it doesn't find a match, ØD is compared with the byte in the table. The ØDs in the table indicate the end of a word. If an ØD is found, the processor branches forward and increments the X counter and the word number in &FE. The same byte is supplied by the CHRGOT and the processor jumps back to the beginning of the routine again, ready to start on the next word in the table.

If no ØD is found, the byte of the table is compared with 1, which indicates the end of the table. If a 1 is found, the processor branches forward into the exit routine at &CFØØ.

If the letter in A does not match one in the table, and the table has not reached the end of a word (or the end of the table), the X counter is incremented and the processor jumps back to check again.

CHECKING THE SPELLING

Once the initial letter of the graphics command has been located, the processor has to check that the rest of the letters of the new keyword match the ones in its table. Otherwise it returns a syntax error.

```
    ORG 49249
FND INX
    LDA #&ØD
    CMP &CFØØ,X
    BEQ &CØ89
    LDA #&Ø1
    CMP &CFØØ,X
    BEQ &CØ89
    JSR &ØØ73
    CMP &CFØØ,X
    BEQ FND
    LDY &FB
    STY &7A
    LDY &FC
    STY &7B
    LDA &FE
    CMP #&16
    BEQ &CØA4
    JMP &CØ46
```

The first thing that is checked for is a ØD or 1 in the next byte of the table. Of course, it won't find either of these on the first pass, but these instructions are part of a loop. When one of these is found, the check is complete and the processor jumps onto the routine that works out which routine it needs to go to.

The next byte of the BASIC program is obtained from the CHRGET routine again and it is compared with the byte in the table. If they match, the processor goes back to the beginning of this routine again and checks the next byte.

If not, the BASIC byte pointer is loaded back into the appropriate system variable. The word number in &FE is compared with 22, &16 in hex, to see whether all the words have been checked. If they have, BEQ &CØA4 branches forward to the exit routine. Otherwise, the processor jumps back to &CØ46, which is marked by the label AGAIN in the routine, to start the checks all over again.

WHICH ROUTINE?

The program now jumps to the correct routine to deal with that graphics command.

```
    ORG 49289
    LDX #&Ø1
    LDY #&ØØ
ADD CPX &FE
    BEQ &CØ97
    INX
    INY
    INY
    JMP ADD
    LDA &CØCØ,Y
    STA &FD
    LDA &CØC1,Y
    STA &FE
    JMP (&ØØFD)
    LDX #&ØB
    JMP (&Ø3ØØ)
```

The X and Y registers are initialized and incremented until the word number in &FE matches the value of X. Then the value of Y calculated is used as an offset to read across the table starting at &CØCØ to find the address of the routine for handling that instruction. The start address of the routine is stored in &FD and &FE.

JMP (&ØØFD) jumps to the routine which starts at the address pointed to by &FD and &FE.

LDX #&ØB and JMP (&Ø3ØØ) make up a small exit routine called when the word in the program does not match any of the command words in the table. The routine pointed to by &Ø3ØØ and &Ø3Ø1 returns the BASIC error message specified by the number that is found in the X register.

Microtip

Long programs

Where *INPUT*'s graphics routines manipulate the screen, BASIC programs longer than 6K will not RUN. So far, all the graphics programs published in *INPUT* have been shorter than 6K. But if you want to write longer ones you will have to move BASIC up memory to the other side of the screen area. You can do that by entering the following POKEs:

```
POKE 43,1:POKE 44,65;POKE 65*256,Ø:
  NEW
```

This will give you 24K free for your BASIC programs.

THE ROUTINE TABLE

Here are the start addresses of the routines.

```
ORG 49344        WOR &CEØØ
WOR &C1ØØ        WOR &CEØØ
WOR &C13Ø        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ        WOR &CEØØ
WOR &CEØØ
```

In this article only two of the graphics commands are going to be covered. The rest will be dealt with in subsequent chapters. So there are only two start addresses of active routines in this table—C13Ø and C1ØØ. For the moment, if any of the other command

words are found, they will be directed to a temporary routine at CEØØ. The start address of the other graphics routines will be filled in when the routines are dealt with.

THE @COLOUR ROUTINE

```
ORG  494Ø8
JSR  &B79B
TXA
STA  &DØ21
JSR  &AEFD
JSR  &B79E
TXA
STA  &DØ2Ø
LDX  &CØØE
JMP  &ØØ79
```

This part of the program makes use of several ROM calls. JSR &B79B jumps to the subroutine at &B79B which inputs the first parameter after the COLOUR command. This is transferred from the X register into A and stored in the input/output location which controls the screen colour.

The ROM routine at &AEFD deals with commas. In this case the comma is ignored and the next parameter is input. This is called at &B79E. The first three bytes of the routine move it onto the next byte, but this is not necessary here as the comma routine has already done it. It is stored in the location that controls the border colour. Then the error status is loaded into X and the processor exits the program via the error message routine.

THE @HIRES ROUTINE

```
       ORG  49456
       LDA  #&2Ø
       STA  &FE
       LDA  #&ØØ
       STA  &FD
ROUND  LDY  #&ØØ
LOOP   STA  (&FD),Y
       INY
       CPY  #&ØØ
       BNE  LOOP
       INC  &FE
       LDX  &FE
       CPX  #&4Ø
       BNE  ROUND
       LDA  #&3B
       STA  &DØ11
       LDA  &DØ18
       ORA  #&Ø8
       STA  &DØ18
       JSR  &B79B
       TXA
       AND  #&ØF
       ASL  A
       ASL  A
       ASL  A
       ASL  A
       STA  &Ø2
       JSR  &AEFD
       JSR  &B79E
       TXA
       AND  #&ØF
       CLC
       ADC  &Ø2
       STA  &Ø2
       LDA  &Ø2
       LDY  #&ØØ
REPEAT STA  &Ø4ØØ,Y
       STA  &Ø4FA,Y
       STA  &Ø5F4,Y
       STA  &Ø6EE,Y
       INY
       CPY  #&FB
       BNE  REPEAT
       LDA  #&6Ø
       STA  &C23E
       JSR  &C2Ø8
       LDA  #&4C
       STA  &C23E
       LDA  #&C8
       STA  &DØ16
       LDX  &CØØE
       JMP  &ØØ79
```

LDA #&3B and STA &DØ11 sets the Vic chip control register bit-map (or @HIRES) mode. Then the next four instructions prepare the Vic memory control register by switching it from 21 to 24 to give hi-res.

The next parameter is then input, switched into the A register and stored in &Ø286—the current character colour code system variable. The following comma is then skipped over and the next parameter is input.

Then the error code condition is loaded back into X and the processor leaves the program again via the error print routine.

Q+A

Will all the graphics programs for the Commodore in *INPUT* work with the graphics command being added to BASIC here?

When you have completed adding the instructions in the last article in the series, provided you add the @ sign before the graphics instructions in your programs, they will all work.

There is one exception though and some alterations have to be made to the program on page 569. These are:

```
1Ø POKE 51,255:POKE 52,31:POKE 55,
   255:POKE 56,31:CLR
5Ø DATA 169,Ø,133,251,133,253,169,
   32,133,252,169,96,133,254,
   16Ø,Ø
6Ø DATA 177,251,145,253,192,63,
   2Ø8,16,165,252,2Ø1,63,2Ø8,1Ø
```

THE NEW WORD ROUTINE

This routine returns the error message.

```
ORG  52736
LDX  #&ØØ
LDA  &CE13,X
JSR  &FFD2
INX
CPX  #&12
BNE  &CEØ2
LDX  &CØØE
JMP  &ØØ79
TXT  'NOT . . . IMPLEMENTED'
```

THE ASCII TABLE

This table carries the ASCII data of all the new keywords. Each word ends with a ØD byte and the table ends with a Ø1.

```
ORG  52992
TXT  'COLOUR'
TXT  'HIRES'
TXT  'MULTI'
TXT  'NRM'
TXT  'LOWCOL'
TXT  'HICOL'
TXT  'PLOT'
TXT  'LINE'
TXT  'BLOCK'
TXT  'PAINT'
TXT  'TEST'
TXT  'CSET'
TXT  'REC'
TXT  'CHAR'
TXT  'TEXT'
TXT  'ARC'
TXT  'ANGL'
TXT  'CIRCLE'
TXT  'DRAW'
TXT  'ROT'
TXT  'FLASH'
TXT  'OFF'
BYT  &Ø1
```

TESTING

Use the following BASIC graphics program to test that your machine code is working:

```
1Ø @HIRES Ø,1
2Ø FOR Z=8192 TO 16191 STEP 5:
   POKE Z,255:NEXT
3Ø FOR Z=Ø TO 15:@COLOUR Z,Z:
   NEXT Z:GOTO 3Ø
```

SYS 49152 calls the whole routine.

DATABASE MANAGEMENT SYSTEMS

The ability of the computer to work at fantastic speeds and to store information in such a way that it takes up little space makes it an ideal tool for searching through and manipulating large collections of information. And one of the most common serious uses of home computers is as a *database management system*, or DBMS.

A database is essentially a collection of one or more datafiles (see pages 622 to 627), themselves little more than a collection of individual records. But the power of the computer means that by embracing usually more than one file, a database assumes rather greater power than a mere electronic equivalent of a filing cabinet.

WHAT IS A DBMS?

The difference lies in the way the information is manipulated. In a datafile, information is simply entered according to the records required—the various entries are established and all the records within the file follow exactly the same form. The same pattern of records could, in practice, extend beyond a single file—a large company may, after all, like to isolate personnel records departmentally.

Left simply in storage, these various datafiles are only of use if and when accessed from outside. And essentially they are little more than electronic versions of paper records at this stage.

The significant difference is that information in a database is specifically set out ready for subsequent work such as sorting, record searching, amendment or replacement of entries, and so on.

In reality, a DBMS is capable of much more, particularly in that details of one file can be used to generate or update information in another.

USES OF DATABASES

Commercially available database management systems are being put to some remarkable uses. A vicar uses one for storing lessons and sermons, a hospital uses a database on a home micro to match up blood types, and they have also been used in an attempt to forecast football results and horse race winners!

DBMS can be used in any application where a great deal of information has to be filed and repeatedly accessed later. And of course it doesn't matter whether the information is business or hobby interests—a single DBMS program can be used for both.

Information can be disgorged in the form of mailing lists, labels, company reports, personalised form letters, invoices, statements, debtors lists, stock lists, and much more.

Although the terms 'database' and 'database management system' are often used, incorrectly, to describe the same thing, there is in fact an important difference. Confusion often arises because the DBMS is sometimes referred to simply as the database, meaning the whole package, hardware and software, rather than by its proper title, a database management system.

And there's a clear difference between information which is simply collected as a file of data and confined to storage—such as a straightforward correspondence file—and information which is filed and then frequently used as a *source* of information (in effect a reference file).

ORDERING FILES

In many instances, a datafile or simple database will have a fixed record format, field size and headings. A true DBMS should give you the flexibility of defining your own records from scratch, accepting, of course, memory and other system limitations.

There must be some order to the database to start off with and defining this is an essential first step in using a DBMS. It need not be highly organised but there must be some sense to it. After all, 'cemoprtu' is an organised collection of information (it's in alphabetical order) but it will not make much sense to man or machine until it is reorganised to become 'computer'.

Similarly a filing cabinet with all the addresses of all the members of a club or company in one file, all their names in another and all their telephone numbers in another would be useless because it would be impossible to tell which name belonged to which address or phone number.

Q+A

Would a purpose-built filing program or a database be better suited to my needs for a simple application such as creating mailing lists?

A purpose-designed program would probably be the better choice. Such a program would be simpler and shorter, and could have special, easily accessed features such as printout of all entries.

But the fields of even a simple DBMS can be defined in a way that would yield a suitable mailing list (where output is in the form of labels). The data may have to be read by an external program unless there is a label facility, only likely with more expensive packages.

FILE MANAGING

Taking again the comparison with a manually run 'paper' office, a DBMS is like the person whose job it is to make sure that information from a whole collection of physically separate but subject-linked files is passed to the appropriate department for attention—the file manager, if you like.

In a manual filing system, the filing manager has to make decisions about how the information should be stored. The file of people who work for a company, for instance, could be arranged alphabetically. But it could also be arranged by length of service, rates of pay, by department or position in the company.

The filing manager's real problems start when other departments want specific information. Someone may want to know how many (and which) staff are over the age of 50, for instance, while someone else may need a geographical breakdown of where staff live. If the files are arranged alphabetically, then the filing manager has no choice but to read through all the record cards to extract the appropriate information from them.

Even if your filing needs are simple now, a proper database management system could be a worthwhile long-term investment. Here's a look at what a DBMS can do

- WHAT IS A DBMS?
- USES OF DATABASES—IDEAL FOR HOBBIES AND BUSINESS
- ORDERING FILES
- FILE MANAGEMENT
- INFORMATION SEARCHES
- DESIGNING A RECORD
- IMPORTANCE OF THE KEYFIELD
- ACCESSING INFORMATION
- DIFFERENT TYPES OF OUTPUT

If this sort of information is needed regularly, the file manager may decide to set up two other files, one stored into age groups and the other organised geographically. It would be more efficient if, instead of complete files, these two extra files took the form of indexes cross-referenced to the main file.

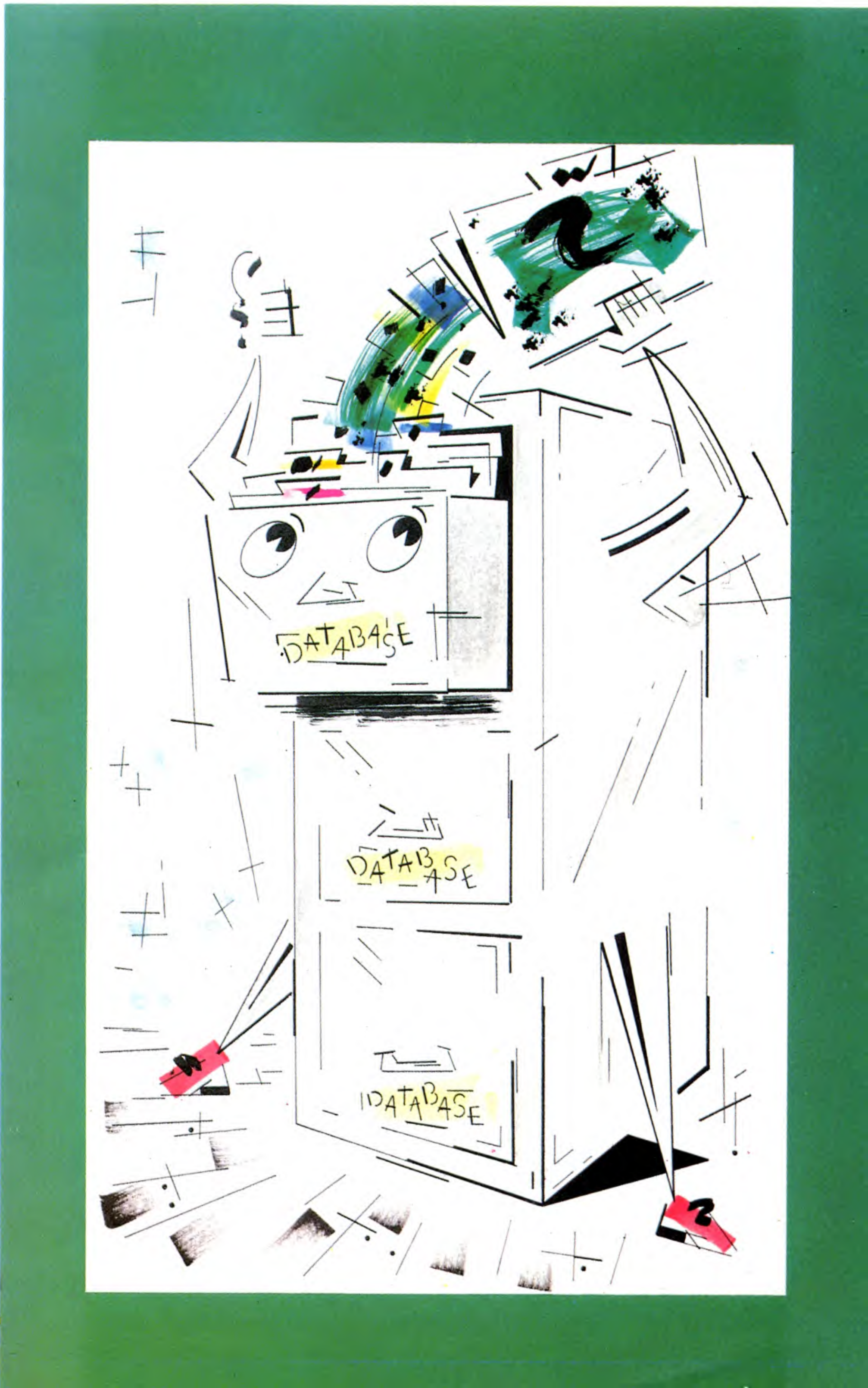
In a computerised system, the choices are exactly the same. A simple DBMS would file information serially in the form of a sequential file. This is like the filing manager using just one file, say the alphabetical file. A more sophisticated DBMS, and most serious commercial software falls into this category, uses relational or relative files to store information. This is equivalent to the filing manager setting up extra files which are cross-referenced to the main file.

Unfortunately for those who use cassette storage, a relative file requires random access—the ability to search anywhere on the file and to move backwards and forwards through it. (Although the words 'backwards and forwards' do not really apply to a disk drive since data is stored in order to make the best use of space on the disk and not necessarily in logical sequence.) Normally tapes are not randomly accessible while disks are.

The type of DBMS used in business is only really practical using a fast and efficient method of storing and retrieving lots of information—which is why business machines require a disk drive. This is not, however, essential and the type of small database you are likely to use at home, an address book for instance, will work efficiently using a cassette drive. But it may be necessary to rewind the tape regularly so that the computer can search through it again.

FILING

Data can be stored using one of several distinct methods and the most common is in the form of a *sequential file* (see pages 622 to 627). While this is the most efficient in its use of available storage space and, because of its serial nature, the only method available to tape users, it imposes certain restrictions on the use of a database. A sequential file is fine if you want *all* the information displayed or output—but it's not much good for any job



Microtip

Choosing your program

A single record within a DBMS can be designed to yield information which could be used for reports, mailing lists, invoices, general accounts, form letters, 'diary' letters, stock updates—and much more. Anticipate your fullest needs when you first get down to the business of choosing a database program. If you want flexibility, it is best to go for the better quality DBMS programs.

Few of the simpler databases permit you to adjust the field lengths at a later date. Still fewer allow you to insert additional fields within existing ones or move the fields around. You can get round this on some systems by selectively importing 'old' information.

that requires isolation of specific details.

A sequential file system is used by many of the more simple datafile or database programs, but can be emulated by more sophisticated packages when the distinct advantages of a sequential file are required, typically for generating *reports* (see below) and output like a non-specific mailing list (in other words, labels printed from all the records in the file, start to finish).

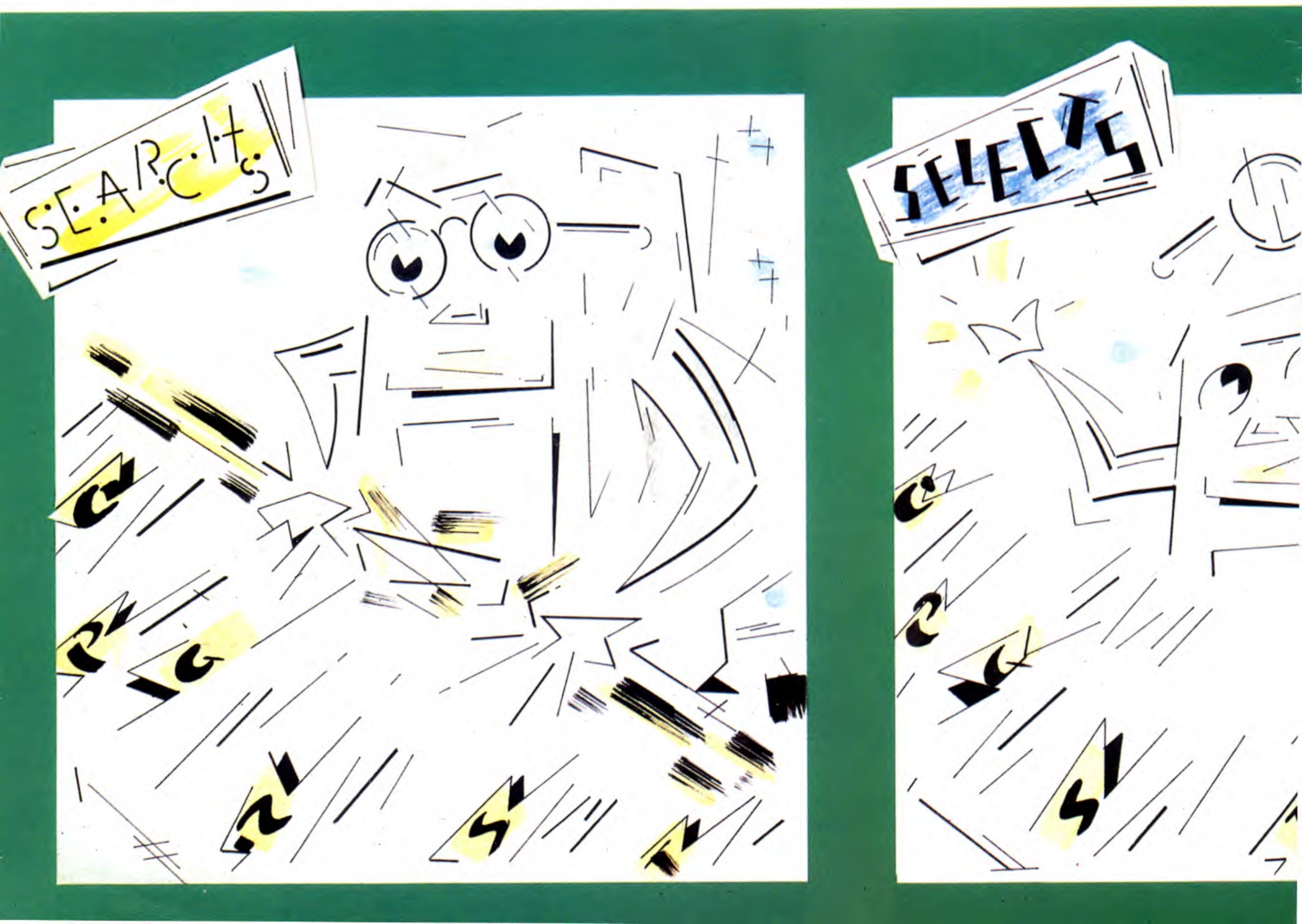
Much more sophisticated *direct access* file handling methods are used on more heavyweight DBMS. From the 'outside' this may not appear to amount to much, but the presence of commands and procedures to enable sophisticated multi-parameter sorts and searches would indicate that a wholly sequential file system is *not* in use. So if information is required—or *possibly* required—for use in or from other programs, do make sure that there's suitable provision for creating the necessary transfer files.

These are often called *export* and *import* files. The 'raw' information within them is separated into fields and records usually only by a commonly recognized symbol, typically the value of a carriage return (CHR$13) but other *field separators* can be used.

Use of predefined field separators enables information to be compacted to make maximum use of the available memory or storage space. Otherwise, with the use of carriage returns alone, a fresh display line is required for each new item of information, regardless of whether it takes a third of the line or all of it. The program originating export files must be able to specify separators which can be recognized by the program which is to import that data.

Using this system information could be *exported* by a DBMS and *imported* by a wordprocessing or spreadsheet program—or vice versa.

Files which are to be read by different programs must of course share a common language—particularly if that same file is to be 'read' by other computers. A conversion facility for the production of standard (or near standard) ASCII sequential files is perhaps an essential requirement in any program used to produce data for transfer by a modem and the telephone network.



INFORMATION SEARCHES

The most efficient method of searching for information also requires a disk drive. The only type of search possible with a cassette unit or Spectrum Microdrive is a linear search, starting at the beginning and working a way through the file testing the 'search key' against each record key until a match—the 'target'—is found. A binary search, on the other hand, starts in the middle.

This may sound illogical but it is, in fact, the way that everyone searches manually for information. Think about looking for a name in the telephone directory—Masters, for instance. Opening the telephone book at the beginning and working our way through the entries until you come to Masters would be very time consuming. So it is normal to open the book in the middle. If if falls open at Jones, then you know that the entry you want is in the second part of the book, on a higher page number. This means you can eliminate all the low pages up to Jones.

The next step is to open the second part of the book in the middle, that is three quarters of the way through the whole book. The entries on this page may be Smith in which case we have gone too far, or 'high'. You want a lower page number so you turn to a page that's halfway between the first page you opened and the second page. If you carry on in the same way you will quickly home in on the correct entry.

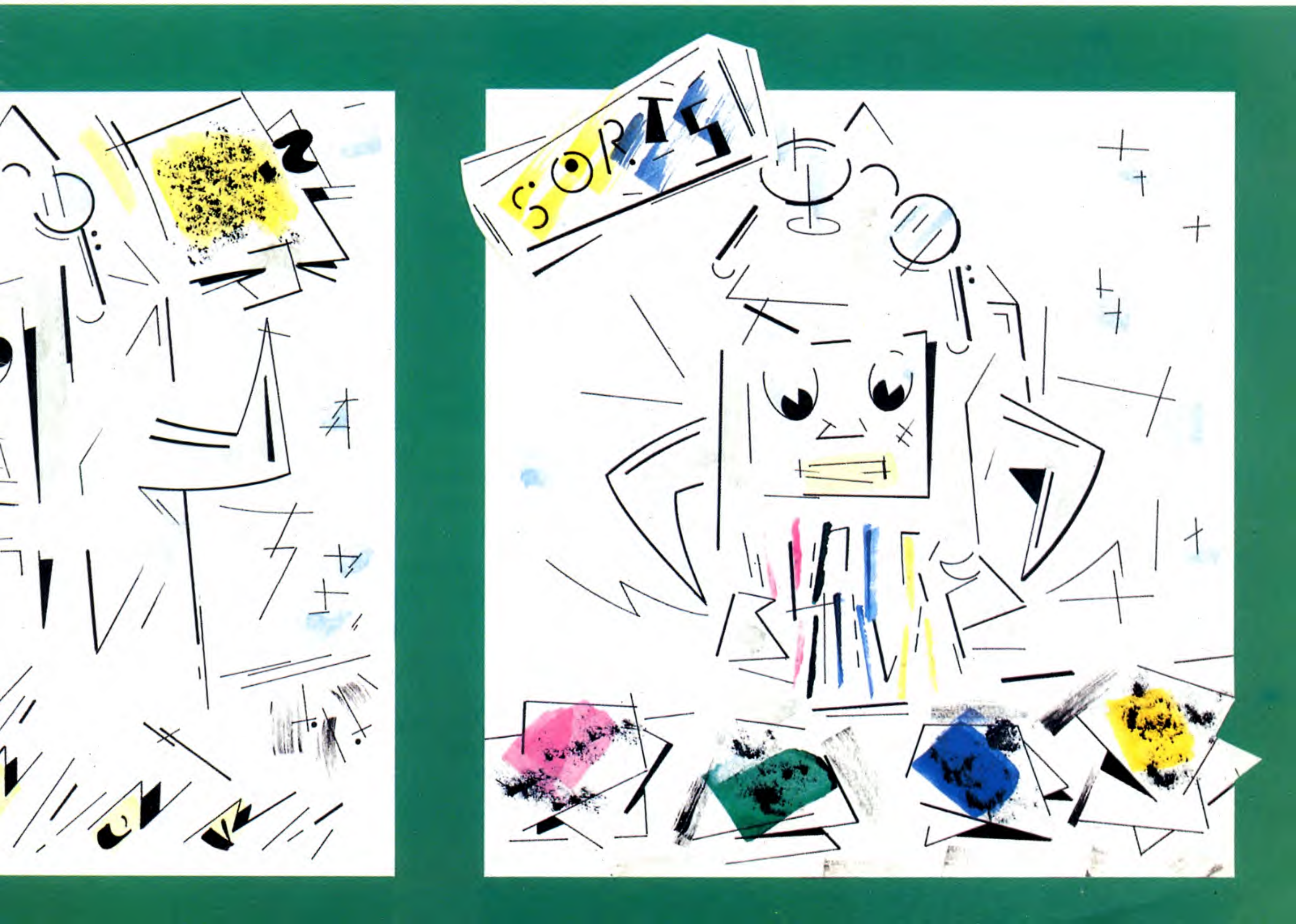
Anyone who has played a 'Guess the Number' game, either with the computer or with a friend, will know the idea and will also be aware of how surprisingly fast and efficient such a search is. The reason why it is called a binary search should be obvious. Each time a choice is made between two alternatives. Are we on a page that's too low (let's give it the value Ø) or are we on a page that's too high (a value of 1)?

If it's a human being that is doing the search, the chances are that at some point they will start guessing at which page the entry will appear. Instead of adhering strictly to the rules we usually go to a page which we think might be close to the entry we want. The computer cannot do this, of course, and must work strictly by logic, according to the rules.

Q+A

What are the procedures for using a typical DBMS?

Defining the record format for a particular file is usually the single major task you face. This involves designing its appearance and content. The fields can take various forms—for example character, numeric, data, calculation—and the right combination is important. A neat display makes it easier for users to find their way round the record, and good planning makes sure that maximum possible use is made of the field lengths, positions and field types. Once the record has been defined, you can proceed with making entries and then use the functions available to you within the program.





SEARCHING AT SPEED

The electronic filing cabinet consists of three main elements: the computer, the data and the program which manages the data. It is much more efficient than any manual system could hope to be. And for one reason only: speed.

Asking a computerised system for a list of all those employees of the company who are aged over 50 requires exactly the same action as in a manual system. The filing manager and the computer search through the file, laying on one side all the appropriate records. But a computer can search through the file at a speed many times faster than a human being—seconds rather than hours.

Searching through a sequential file (which is the sort of file stored on a cassette) is slower than searching through relative files for both the computerised and the manual systems. This is because every record in a sequential file has to be looked at one by one whereas a relative file will usually have a separate index for every separate item of information or 'field' on the records. Searching through the index containing dates of birth will obviously be much quicker than searching through every individual record. This is true whether the program utilises a linear search or binary search. A binary search through relative files will obviously be quickest of all.

DESIGNING A RECORD

There are many commercial DBMS programs available for most makes of home micro but writing your own program for something like a simple address book is not too difficult (see the file program on pages 46–53 and 75–79). Whether you are writing your own program or using commercial software it is essential to know how a DBMS is constructed!

Designing the format of a record and defining the content of each field of that record are the first steps of using a DBMS. A full-blown DBMS will enable you to design the record from scratch and you may be presented with little more than a blank screen which you have to *format* according to your requirements. On more simple systems the position of the fields which go to make up the record may already be defined—likewise the size and position of the entry sections. In this case you may be required to do little more than enter the number of fields required, and follow this with a heading for each of the fields. The file program on pages 46–53 and 75–79 operates in this way.

For those systems which allow you to format the record precisely to your requirements, the blank screen can be regarded exactly like a blank sheet of paper. The first thing you need to do is to decide what you want to write on the paper!

Aside from the cosmetic appearance of the record (you can usually incorporate graphic symbols and lines in a record definition of this type), you must decide which items or fields will be referred to regularly.

A field can be regarded as a heading for the information. A personnel file for a company may contain the following information: Name, Address; Date of Birth; Sex; Marital Status; Date started employment; Position; Salary. Each one of these headings is known as a field. Great care must be taken when establishing the number and size of each field since they cannot usually be added to or amended later. Obviously, the fields on every record in a file must be the same and it's often useful to specify one extra field, perhaps headed 'Notes', even if this field is left empty on most records.

The information within each field on a record card can, of course, be changed but the fields themselves must remain the same.

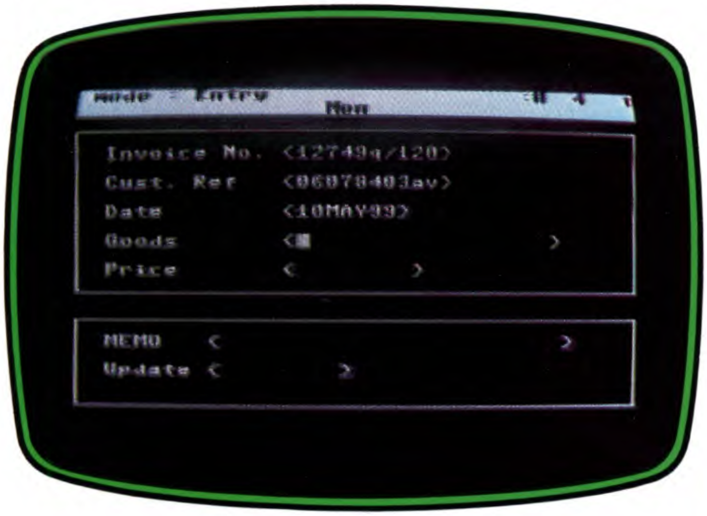
Name, Address, Date of Birth, Telephone Number are obvious fields for an address book. But you might decide that part of the address, the town, for instance, is an important piece of information. In this case you could specify the town as a separate field.

Fields are especially important in relative files, although these are only really practical for those with disk drives. The information in each field can be stored in a separate index, and each entry in the index will contain a pointer to the complete record in the main file. All the telephone numbers in your address file will be contained in an index headed, appropriately, Telephone Numbers. And each number will be prefaced or followed by a unique code which indicates the record in the main file.

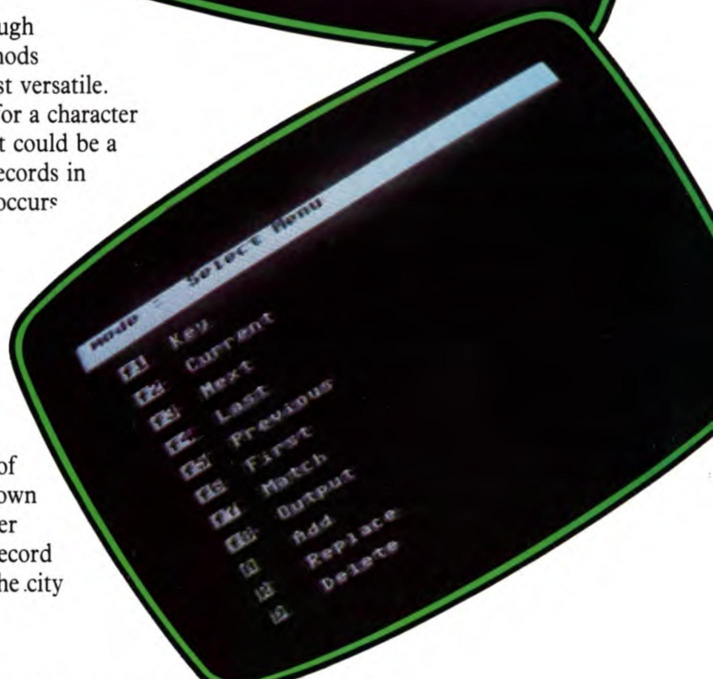
Serious commercial database management programs write the separate indexes and carry out all the necessary cross-referencing automatically every time information is saved.

This is why, with a good DBMS, we can say: 'Find all people named Smith who live in Birmingham and were born before 4/2/1952 then print out their telephone numbers.'

The picture shows various screens of a commercial DBMS (Superbase 64 for the Commodore 64). The first is one of two general menus which outline the main functions of the program. Defining a record ('format') is one of these (next picture). Record appearance and field name and types can be specified. The next picture shows an entry being made. Next is shown the program option, a feature which can be used to generate tailor-made applications from within the main program. Finally there's one of the secondary menus







THE KEYFIELD

The *keyfield* is the most important field in a record—the item of information you will need access to most frequently. It is this at which the DBMS will look when sorting records into order, or if a general search is ordered (this is the quickest way of accessing information). So the keyfield must be the one item of information which is most memorable and which is, as far as possible, unique to each record. The keyfield in a personnel record for instance is likely to be the surname.

ACCESSING INFORMATION

Once the information has been stored, you need to get access to it. The more sophisticated a DBMS, the more flexible it will be and the easier it will be to retrieve exactly the information you want. There are a number of standard methods of searching for information, some of which have been mentioned previously.

A *keyfield search* simply runs through the records (in memory) and stops to display the record which matches the search criteria.

If a system of *indexed* fields is used, each field is filed in its own field index. In the company personnel file example all dates of birth would be in a separate index and each entry would point to the record in the main file.

A *criteria search* can hunt through information in one or more fields. You could find out, for example, how many of your employees were male, over 60 and earned above a certain given level. In this instance the computer would need to conduct three searches more or less simultaneously.

Finally there's the standard *character string search*. Although the slowest of all search methods used on DBMS, it is the most versatile. The computer simply looks for a character string—usually a word, but it could be a number—and gives you all records in which that word or number occurs. It is very useful for gaining access to information which we have not specified as a field. So in an address book file, even though the whole address might be specified as a field and the town or city is not, you can find out which of your friends live in a given town or city by asking the computer to carry out a search for all record cards in which the name of the city or town occurs.

OUTPUT

Hard-copy output in the form of labels, lists and reports is really what DBMS is all about. Thinking about exactly what you want in this respect obviously should influence the very structure of the records which form the files of the DBMS. But such is the flexibility of some systems, that information can be plucked from all over the place in a record and presented in almost any desired position and order when it comes to the printed output.

The simplest form of output is in the form of what is called a *report*. On selecting this option you are usually asked by the core program for the fields which are required. Records are then printed out in strictly sequential order. Some DBMS may even allow you to choose which records will be printed.

Other types of printout, typically for mailing lists, invoices and other 'form' printing require a measure of formatting and special instructions may be provided by the DBMS program for you to do this.

ADAPTING YOUR UDGS

Is there something missing from your UDG generator? This article provides extra program lines to give you even more scope to design exciting UDGs

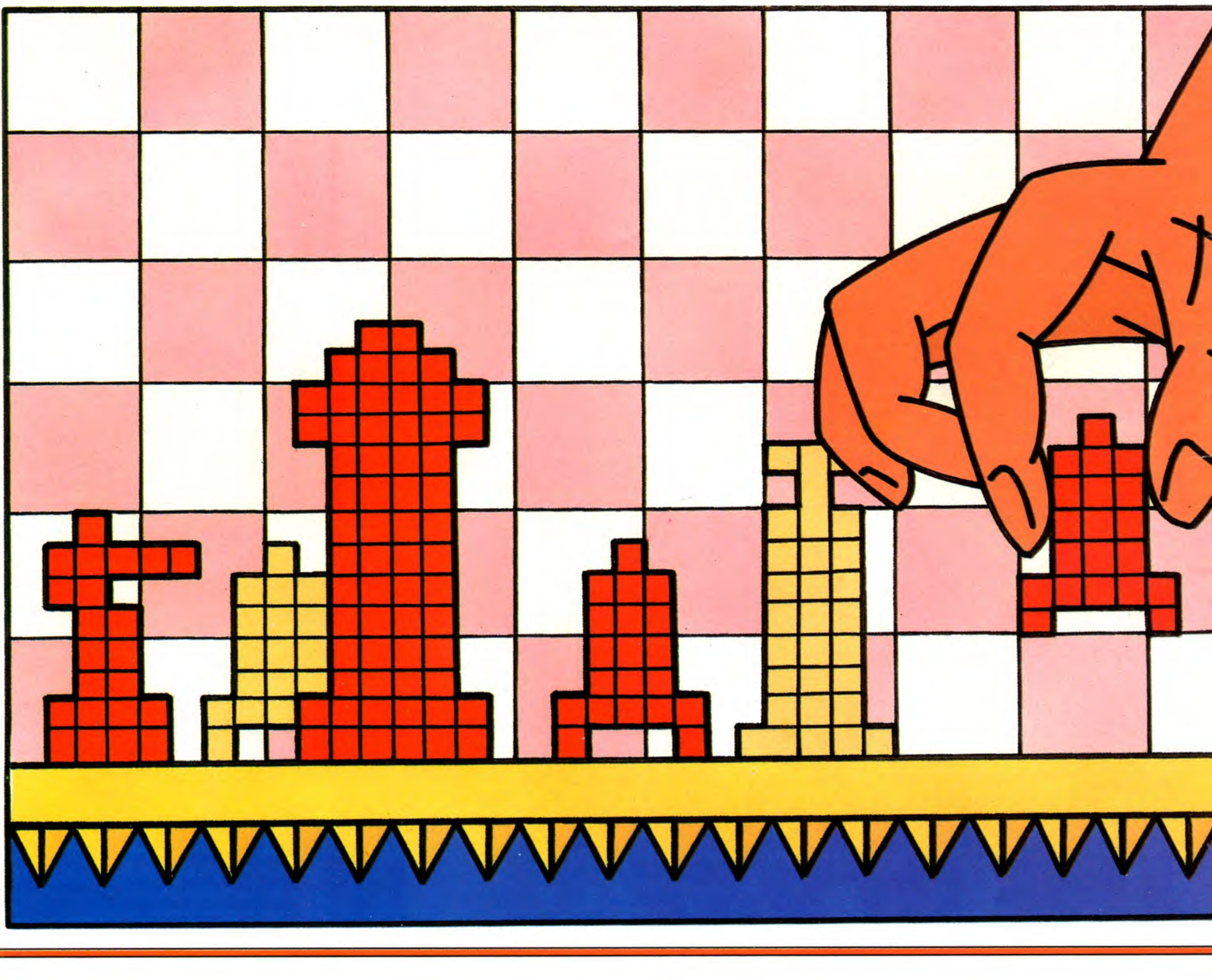
This article provides the rest of the character generator programs. The new program lines add several more editing facilities to make it even easier to design your own graphics characters. There is also a guide to using the UDGs you have designed, by calling them up in other programs.

When you add the new lines to the program given in the last article, six new functions become available to you.

You will see the first function as soon as you RUN the new program. It shows the values of the eight bytes of the character in the grid, and an actual size version of it on the screen display. These bytes are added by a short machine code routine, which updates both the numbers and the actual size UDG every time you set a new point under the cursor.

The new program also lets you clear the grid simply by pressing C to save you having to delete every pixel.

There are three more control keys which you can use to alter the UDG in the grid. If you press M the UDG is mirrored from left to right. This means that whenever you want to design two UDGs to make up one symmetrical figure (a spaceship, for example) you can start by designing one of the UDGs, and store the finished version. You can then call this back and get the computer to create the second half for you. The mirror function is also useful if you want two figures, one facing left, one facing right.

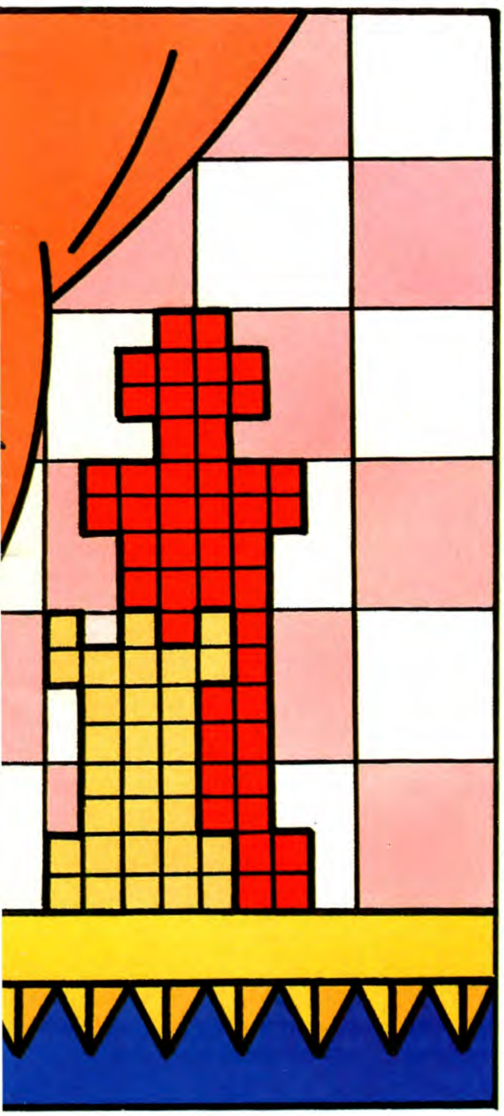


- NEW CONTROLS
- CHECKING THE DATA
- INVERSE UDGS
- REFLECT YOUR DESIGNS
- ROTATING UDGS
- CLEARING THE GRID
- PRINTING OUT THE DATA
- USING THE UDGS IN YOUR OWN PROGRAMS
- LOADING UDGS FROM TAPE

Similarly, you can rotate your UDG through 90 degrees by pressing the R key. This facility is useful, since you can design just one UDG—say, a rocket—and then rotate it to create UDGs of a rocket being fired in any of four different directions.

The last of these three control keys, the I key, gives you the inverse of the character. Press it twice and it gives you the inverse of the inverse—to give you the original character back again.

There is also a printer routine, which you can use to print either the DATA values for each character in the UDG bank, or a screen dump. The screen dump may not work if you have an independent make of printer, but don't worry—a future article in *INPUT* will show how to get your printer to produce a screen dump when this facility is available.



Press Z to activate the printer routine, and then either D or S. D prints out the DATA, while S produces a screen dump. If you press any other key after the Z, the computer returns to the main loop of the program.

If you do not intend to use a printer, you need not type in Lines 257Ø to 259Ø. You should, though, add this line:

```
257Ø GOTO 2ØØØ
```

Here are the extra lines:

```
5 CLEAR 31999
12 LET T=Ø: FOR N=32ØØØ TO 32227:
   READ A: POKE N,A: LET T=T+A: NEXT
   N: IF T<>21691 THEN PRINT FLASH 1;
   "ERROR IN DATA": STOP
2Ø2Ø PRINT AT 1Ø,21;CHR$ 139;CHR$
   131;CHR$ 135;AT 11,21;CHR$ 138;AT
   11,23;CHR$ 133;AT 12,21;CHR$ 142;CHR$
   14Ø;CHR$ 141
2Ø3Ø RANDOMIZE USR 32ØØØ
253Ø IF INKEY$="I" THEN RANDOMIZE
   USR 32Ø92
254Ø IF INKEY$="C" THEN POKE 321Ø6,Ø:
   RANDOMIZE USR 32Ø92: POKE 321Ø6,12
255Ø IF INKEY$="M" THEN RANDOMIZE
   USR 32145
256Ø IF INKEY$="R" THEN RANDOMIZE
   USR 32183
257Ø IF INKEY$<>"Z" THEN GOTO 2ØØØ
2575 INPUT "S(CREEN DUMP) OR
   D(ATA)?□"; LINE Z$
258Ø IF Z$<>"S" AND Z$<>"D" THEN
   GOTO 2ØØØ
259Ø IF Z$="S" THEN COPY : GOTO 2ØØØ
26ØØ LET CH=65: FOR N=USR "A" TO
   USR "U"+7 STEP 8
261Ø LET TA=Ø: LPRINT CHR$ CH: FOR
   M=N TO N+7: LPRINT TAB TA;PEEK M;:
   LET TA=TA+4: NEXT M
262Ø LPRINT : LET CH=CH+1: NEXT N
91ØØ DATA 62,2,2Ø5,1,22,62,22,215,62,
   8,215,175,215,33,11,72,221,33,118,72
911Ø DATA 6,8,197,6,8,14,128,175,5Ø,
   91,125,126,254,1,4Ø,7,58,91,125,
   129
912Ø DATA 5Ø,91,125,2Ø3,57,35,16,239,
   58,91,125,221,119,Ø,229,221,229,
   62,23,215
913Ø DATA 62,5,215,33,9Ø,125,2Ø5,4Ø,
   26,62,13,215,221,225,225,17,24,
   Ø,25,221
914Ø DATA 229,2Ø9,2Ø,213,221,225,193,
   16,189,2Ø1,Ø,Ø,33,11,72,6,8,197,
   6,8
915Ø DATA 197,126,254,1,229,4Ø,12,6,7,
   62,1,119,36,16,252,54,255,24,13,6
916Ø DATA 4,54,85,36,54,171,36,16,248,
   37,54,255,225,193,35,16,219,17,
   24,Ø
917Ø DATA 25,193,16,2Ø9,2Ø1,33,11,72,
   6,8,197,6,4,17,7,Ø,197,229,126,25
918Ø DATA 78,119,225,113,35,27,27,193,
   16,242,17,28,Ø,25,193,16,229,
   2Ø5,92,125
919Ø DATA 195,92,125,221,33,11,74,33,
   235,72,6,8,197,6,8,197,221,126,
   Ø,119
92ØØ DATA 221,35,6,32,43,16,253,193,16,
   241,17,24,Ø,221,25,17,1,1,25,
   193
921Ø DATA 16,226,2Ø5,92,125,195,92,125
```

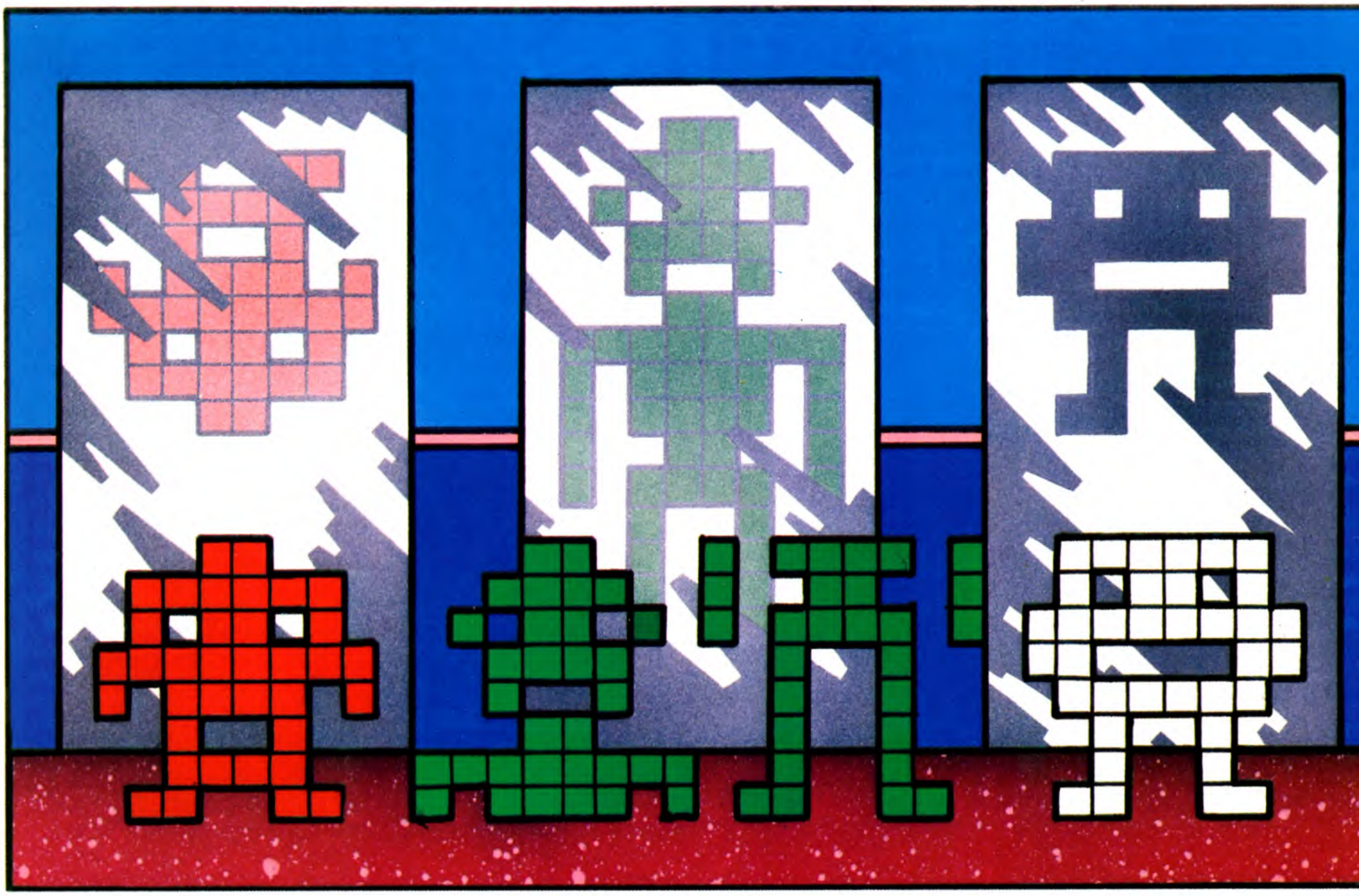
The DATA at the end of the program is POKEd into memory and is machine code. There are three separate routines to rotate, mirror and display the actual size UDG (and its bytes) on the screen all the time.

You must take extreme care to type in these numbers accurately. Any error in your machine code will result in the program failing to work properly—or at worst, crashing completely.

There is a check built-in to the program, so that the program will stop if you have typed in the DATA incorrectly. This means that your program should not suffer if you have made an error. You will just be prompted to recheck the DATA.

USING YOUR UDGs

The first part of the program included a SAVE and LOAD option, so that you could make a permanent record of your characters on tape.



By careful selection of what is stored on tape, you can use this program to create several banks of UDGs.

The Spectrum can access up to 21 UDGs at any one time. If you want to use more than this, you have to have either several banks, or redefine the whole character set. The article on pages 450 to 457 explains how you can do this. There is also a guide to how you can call the UDGs up from the bank and into your program.

You can define all the UDGs you are likely to want using the *INPUT* UDG generator, and you can use them all in one program by changing the UDG pointer.

The program assumes that the UDG pointer is pointing to the address you want it to indicate, and so it SAVEs its bytes from this address, and LOADs them back to this address.

When you want to use the bytes for your own characters in your own programs, you can LOAD them back into memory to any address you specify. This means that if you want to have three banks of UDGs in memory, all you need to do is LOAD each bank into memory in different places.

Each bank is 168 bytes long, so you should remember to LOAD each one into areas of memory at least 168 bytes apart, otherwise one bank might erase part of another. To LOAD a block of memory to any address, you can use this command:

```
LOAD "" CODE (start address)
```

No matter what address the block was SAVEd from, you can LOAD it back to any area in RAM.

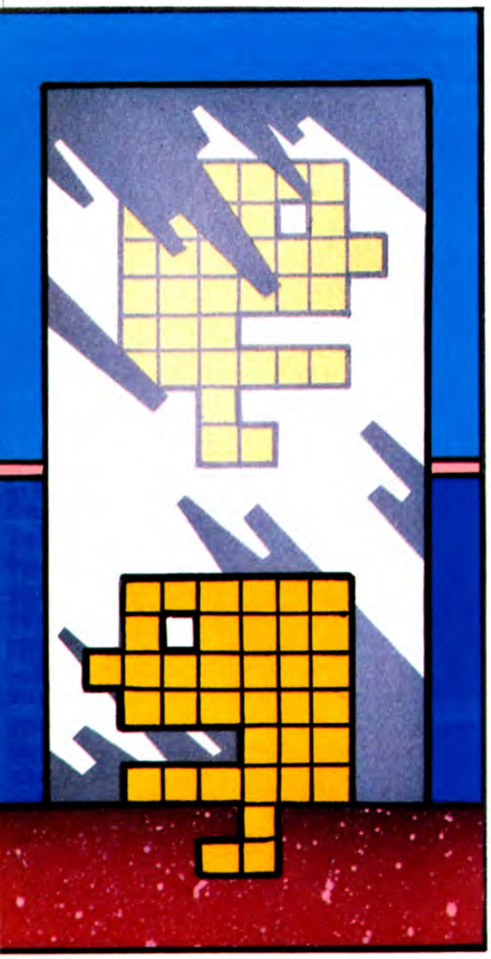
Once you have LOADed each block of memory for the various banks of UDGs you have SAVEd, you can use each one by changing the UDG pointer. The article on pages 450 to 457 explains how you can do this.

Whenever you do use more than one bank of characters in your programs, you should use a CLEAR command to protect the block of memory used for the UDGs from being corrupted by the computer—this, too, is explained in the article on pages 450 to 457.

The program lines below add a number of extra features to the character generator started last time.

If you SAVEd the last program on tape, you can LOAD it back in again. If you did not, type in the lines below and RUN the program.

```
33 FOR Z=Ø TO 7:POKE 12288+
   Z,255:NEXT Z
123 PRINT "■■■■■■■■
   ■■";:FOR Z=1 TO 31:
   PRINT "⊟";:NEXT Z:
   PRINT "■"
125 PRINT"■■■■■■■@□■
   @□■@□■@□■@□■
   @□■@□π@□♠@□
   ■@□⊠@□●@□♣@
   □■@□◆@□■@□■■■"
128 PRINT "■■■■■";:
   FOR Z=1 TO 31:PRINT "⊟";:
   NEXT Z:PRINT "■"
165 IF P=55 AND Y<7 THEN Y=Y+1
210 IF P=4 THEN 1ØØØ
220 IF P=5 THEN FL=1:GOTO 11ØØ
230 IF P=6 THEN FL=2:GOTO 11ØØ
240 IF P=3 THEN FL=3:GOTO 11ØØ
245 IF P=51 THEN FL=4:GOTO 112Ø
260 IF P=49 THEN A=Ø:GOTO 123Ø
270 IF P=54 THEN 13ØØ
1ØØØ FOR Z=Ø TO 7:FOR ZZ=Ø TO7
```



```
1010 C1=1396+ZZ+Z*40:
   C2=PEEK(C1)
1020 IF C2=207 THEN POKE C1,230
1030 IF C2=230 THEN POKE C1,207
1040 NEXT ZZ,Z:GOTO 130
1100 FOR Z=Ø TO 7:FOR ZZ=Ø TO7
1110 C1=1396+ZZ+Z*40:
   C(Z,ZZ)=PEEK(C1):
   NEXT ZZ,Z
1120 FOR Z=Ø TO 7:FOR ZZ=Ø TO7
1130 IF FL=1 THEN POKE 1396+
   ZZ+Z*40,C(ZZ,7-Z)
1132 IF FL=2 THEN POKE 1396+
   ZZ+Z*40,C(7-ZZ,Z)
1134 IF FL=3 THEN POKE 1396+
   (7-ZZ)+Z*40,C(Z,ZZ)
1136 IF FL=4 THEN POKE 1396+
   ZZ+Z*40,207
1140 NEXT ZZ,Z
1145 IF FL=4 THEN A=Ø:
   GOTO 1230
1150 GOTO 130
1230 PRINT "■■■■■■■■■"
1270 PRINT "■";SPC(22);
   Z+1;"■□□□□■■■■■■■";T
1300 PRINT "■■■■■";
   TAB(9);"(D)ATA OR (C)HARACTER?": FOR
   Z=1 TO 25: NEXT Z
1301 PRINT "■■■■■";
   TAB(9);"□□□□□□□□□□□
   □□□□□□□□□□□□□"
1302 GET A$:IF A$<>"D" AND
   A$<>"C" THEN 1300
1303 OPEN 4,4:CMD4:IF A$="D" THEN
   1700
1304 FOR Z=1 TO 26:PRINT SPC(6);
   CHR$(64+Z): FOR ZZ=Ø TO 7: PRINT
   ZZ+1;
1305 A=PEEK(12288+Z*8+ZZ)
1310 AA=A:FOR Z1=Ø TO 7:IF
   A-Z(Z1)=>Ø THEN A=A-Z(Z1):
   PRINT"■□■";: GOTO 1320
1315 PRINT ".";
1320 NEXT Z1:PRINT AA:NEXT ZZ:
   PRINT:NEXT Z
1330 PRINT#4:CLOSE4:GOTO 130
1620 NEXT Z1,ZZ: GOTO 1230
1700 FOR Z=1 TO 26:PRINT CHR$
   (64+Z);"=";:FOR ZZ=Ø TO 7
1710 PRINT PEEK(12288+Z*8+ZZ);:
   NEXT ZZ:PRINT:NEXT Z
1720 PRINT#4:CLOSE4:GOTO 130
```

With an expanded Vic, delete Lines 13Ø1, 13Ø2, 13Ø4, 13Ø5, 131Ø, 1315 and 132Ø, and enter:

```
1300 REM
```

C64

```
125 PRINT"■■■■■■■■■
   ■■@□■@□■@□■
   @□■@□■@□■@□π
   @□■"
210 IF P=39 THEN 1000
220 IF P=47 THEN FL=1:GOTO 1100
230 IF P=55 THEN FL=2:GOTO 1100
240 IF P=63 THEN FL=3:GOTO 1100
245 IF P=62 THEN FL=4:GOTO 1120
260 IF P=14 THEN A=Ø:GOTO 1230
270 IF P=54 THEN 1300
1000 FOR Z=Ø TO 7:FOR ZZ=Ø TO7
1010 C1=7880+ZZ+Z*22:
   C2=PEEK(C1)
1020 IF C2=207 THEN POKE C1,230
1030 IF C2=230 THEN POKE C1,207
1040 NEXT ZZ,Z:GOTO 130
1100 FOR Z=Ø TO 7:FOR ZZ=Ø TO7
1110 C1=7880+ZZ+Z*22:C(Z,ZZ)=
   PEEK(C1):NEXT ZZ,Z
1120 FOR Z=Ø TO 7:FOR ZZ=Ø TO7
1130 IF FL=1 THEN POKE 7880+
   ZZ+Z*22,C(ZZ,7-Z)
1132 IF FL=2 THEN POKE 7880+
   ZZ+Z*22,C(7-ZZ,Z)
1134 IF FL=3 THEN POKE 7880+
   (7-ZZ)+Z*22,C(Z,ZZ)
1136 IF FL=4 THEN POKE 7880+
   ZZ+Z*22,207
1140 NEXT ZZ,Z
1145 IF FL=4 THEN A=Ø:
   GOTO 1230
1150 GOTO 130
1230 PRINT "■■■■■■■■■"
1270 PRINT "■";SPC(12);
   Z+1;"■□□□□■■■■■■■";T
1300 PRINT "■■■■■(D)ATA OR
   (C)HARACTER?":FOR Z=1 TO 25:NEXT Z
1301 PRINT "■■■■■□□□□□
   □□□□□□□□□□□□□□
   □□□"
1302 GET A$:IF A$<>"D" AND
   A$<>"C" THEN 1300
1303 OPEN 4,4:CMD4:IF A$="D" THEN
   1700
1304 FOR Z=1 TO 22:PRINT SPC(6);
   CHR$(64+Z):FOR ZZ=Ø TO 7:
   PRINT ZZ+1;
1305 A=PEEK(7168+Z*8+ZZ)
1310 AA=A:FOR Z1=Ø TO 7: IF
   A-Z(Z1)=>Ø THEN A=A-Z(Z1):
   PRINT"■□■";:
   GOTO 1320
1315 PRINT ".";
1320 NEXT Z1:PRINT AA:NEXT ZZ:
   PRINT:NEXT Z
1330 PRINT#4:CLOSE4:
   GOTO 130
1700 FOR Z=1 TO 22:PRINT CHR$
   (64+Z);"=";:FOR ZZ=Ø TO 7
1710 PRINT PEEK(7168+Z*8+ZZ);:
   NEXT ZZ:PRINT:NEXT Z
1720 PRINT#4:CLOSE4:
   GOTO 130
```

When you have typed in the extra lines, your character generator is complete. You can now move around the grid, setting pixels on or off, and rotate, mirror, invert and clear the grid, too. The function keys at the right of the keyboard are used for several of the new controls.

To rotate the grid left, press F3, and for right press F5. The ability to rotate a single UDG comes in handy when you want to have the same, or very similar, characters able to point in four different directions, such as a spaceship which you can turn round to fire at any point on the screen.

The mirror routine, obtained by pressing F7, mirrors the grid from left to right about an imaginary axis: a vertical line in the centre of the grid. You can use this to produce a symmetrical figure which comprises two UDGs—all you need to do is design one of the UDGs, store it, and then call it back and mirror it to get the other half. Missile bases are ideal subjects for this, as they usually take up two UDGs, and are symmetrical.

You can invert your design by pressing the

F1 key. This turns every pixel to its opposite—on, if it was off, or off, if it was on. This can give quite surprising results.

If you decide, after designing a UDG in the grid, that it is nothing like what you want, you can wipe it out completely, clearing the grid ready for you to start again. Do this by pressing the CLR/HOME key. Because this is at the top of the keyboard, away from the movement keys, you are unlikely to hit it by accident.

USING YOUR UDGs

The first part of the program provided a SAVE to tape routine, so that you could store the UDGs for use in your own programs. You might also like to use the DATA values for each design. You can copy the DATA from the screen by hand by pressing the asterisk (*) key for each UDG in the grid, but this can take a long time. If you have a printer, you can also have the DATA printed out on that. To do this, press the up-arrow key (not the cursor up).

If you want to SAVE the characters onto tape and use them in your own programs without typing in the DATA, you must LOAD them back again. Unfortunately, this is less easy than you might think and you should use the LOAD routine in Lines 141Ø to 142Ø.

Don't forget that in Line 141Ø, you must only use the half of the line which applies to you: the first half if you are using tape, the second half if you are using disk. In this line, you should replace the variable OU with Ø, and the string variable N$ with the name of the stored file (you type in this name when you SAVE the character set from the UDG generator).

By careful planning of your computer's memory, you can have several sets of characters stored in the computer at one time. You can change the area in memory that the DATA is LOADed back into to help you here, so that you can SAVE several blocks of characters from the UDG generator, and then LOAD them all back into memory.

To do this, all you have to do is to change the numbers in the FOR . . . NEXT loop in Line 1413. The first of the two numbers is where the characters start in memory, and the second number is where they end. You also have to set the character set pointers accordingly by changing the POKEs in Line 1Ø. This is explained in the article on pages 450 to 457.

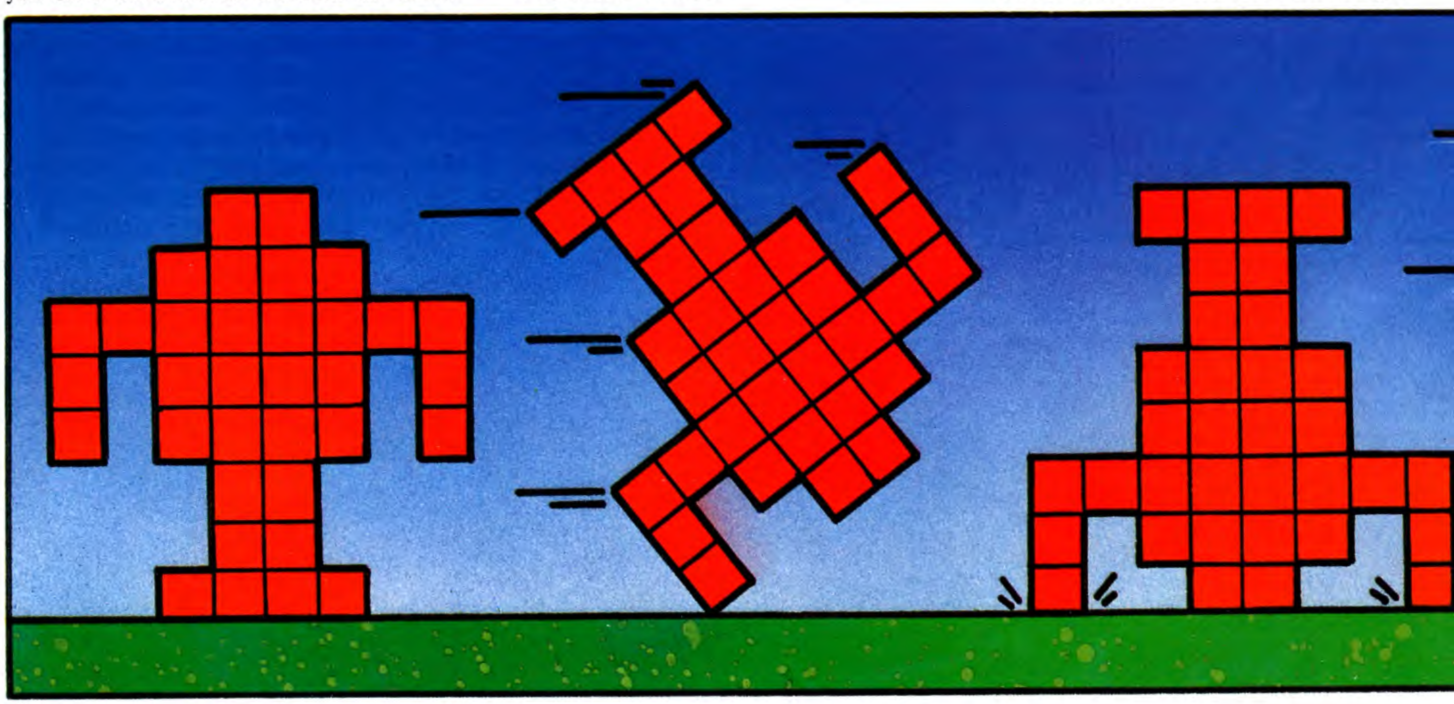
With this method, you can call up a large number of UDGs into your programs, and never have to type in any DATA values for them.

You can extend your character generator using the extra program lines given below. You should first type in, or LOAD from tape, the last part of the program—the new lines do not work on their own.

```
240 IF A$=CHR$(9) THEN
   PROCINVERT:ENDPROC
250 IF A$=CHR$(18) THEN
   PROCREFLECT:ENDPROC
260 IF A$=CHR$(15) THEN CALL
   ROT:PROCNOS:GOTO 360
270 IF A$=CHR$(16) THEN
   PROCPRT:ENDPROC
280 TY=Y
340 PRINTTAB(25,16+TY)STRING$
   (3-LEN(STR$(TY?&C18)),"□");
   TY?&C18
410 FOR T=&C18 TO &C1F:PRINT
   TAB(25,16+T-&C18)STRING$
   (3-LEN(STR$(?T)),"□");?T:
   NEXT:CALL MC:VDU5:MOVE
   (16+X)*32,1023-(16+Y)*32:
   VDU224,4:ENDPROC
700 DEF PROCINVERT
710 FOR T=&C18 TO &C1F:
   ?T=255-?T:NEXT:PROCNOS:
   ENDPROC
720 DEF PROCREFLECT
730 FOR T=&C18 TO &C1F:?&70=0:
   FOR P=0 TO 7: IF ?T AND 2^P THEN
   ?&70=?&70 OR 2^(7-P)
740 NEXT:?T=?&70:NEXT:
   PROCNOS:ENDPROC
750 DEF PROCPRT
760 PRINTTAB(0,28)"PRINTING NOW":
   COLOUR0:VDU2:PROCMOVEUDG(1):
   FOR P=0 TO 31:PRINTTAB(0,29)
   "VDU23, CHARACTER NUMBER";:
   FOR T=&C00+8*P TO &C07+8*P:
   PRINT",";?T;:NEXT:PRINT:NEXT:
   PROCMOVEUDG(6):VDU3:COLOUR3:
   PRINTTAB(0,28)STRING$(12,"□"):
   ENDPROC
```

Once you have added these extra lines, the program makes it even easier for you to design your characters, allowing you to invert, mirror and rotate your UDG and print out the DATA.

There are often several UDGs which you



might want for a program, which are all fairly similar. When this is the case, you have to design almost the same thing several times. The new features of the program help to make this quicker.

The Acorn has no INVERSE command unlike some other computers so, while you can get round the problem by changing the colours, it is often useful to have some inverse characters in memory. For this reason, the UDG generator now has an inverse command. When you press the CTRL and I keys, the computer gives you the inverse of your design: every pixel that was set, or coloured-in, becomes clear, and each one that was clear becomes set.

Another feature which manipulates your design is the reflect facility. Pressing the CTRL and R keys mirrors the UDG in the grid from left to right through an imaginary axis—a vertical line in the centre of the grid.

This facility is very useful for symmetrical designs which take up more than one UDG. Suppose you want to design a laser base for your latest space game; the chances are that it will take up two UDGs, side by side. If the design is symmetrical, you only need design one UDG. When you are satisfied you can Store it in the bank, and then reflect it to get the computer to produce the other half.

Similarly there is a rotate command. The CTRL and O keys rotate your design through 90 degrees. You can use it to produce versions of your UDG pointing in four directions, which you might need, for example, if you are designing rockets for an asteroids game.

The last feature that the new lines add is a printer option. This prints out the eight bytes of DATA for each UDG in the bank to a printer, if you have one. The bytes are printed as VDU statements, with eight bytes after each VDU 23, CHARACTER NUMBER, to make it easier for you to understand. To use this option, press the CTRL and P keys.

If you do not want to use a printer, you should miss out Lines 27Ø, 75Ø and 76Ø. This prevents the computer trying to send information to a non-existent printer.

USING YOUR UDGs

Although you can get the DATA values printed out, you do not *have* to type in each byte for every UDG in order to use them in your own programs—you can just LOAD them in from tape.

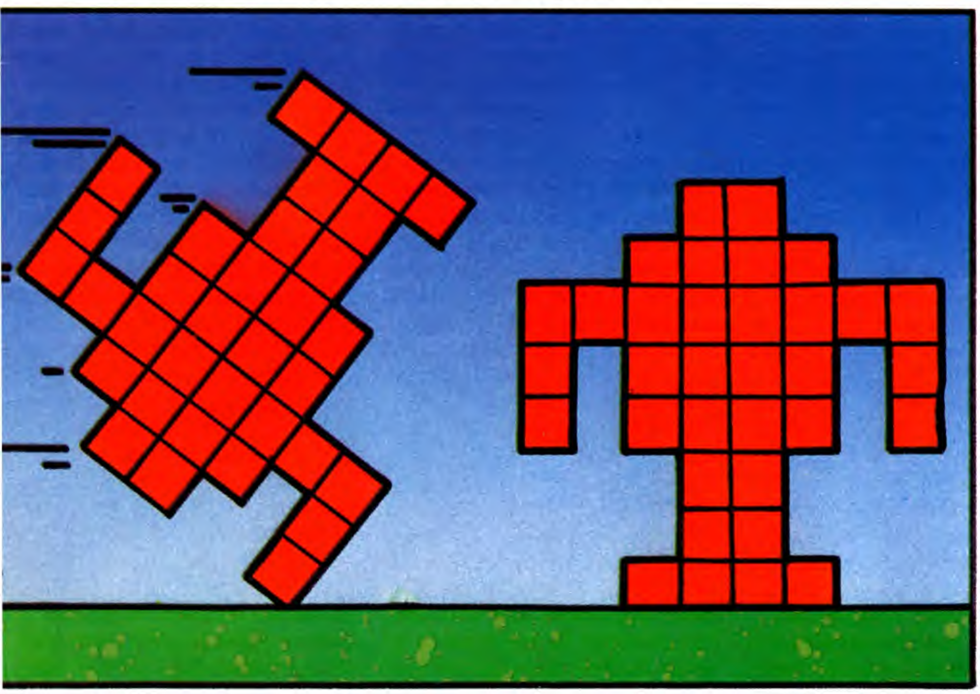
When you SAVE a bank of UDGs to tape with the SAVE option in the program, the computer stores a block of memory on tape. So when you want to put it back into the computer, you have to tell the computer to expect a block of memory instead of a program.

You do this with the command *LOAD, followed by quotes, followed by a number in hexadecimal; this number is the first address that the block is LOADed into.

So, by working out where in memory you are going to store the UDGs, you can fit in as many banks as you want—the only limit is your computer's memory. All you have to do is *LOAD each bank to a different address in memory, so that the new banks do not wipe out the old. The article on pages 450 to 457 explains this.

You do not need to give any address at all after the *LOAD command. If you don't, the block is automatically LOADed back to the area it came from.

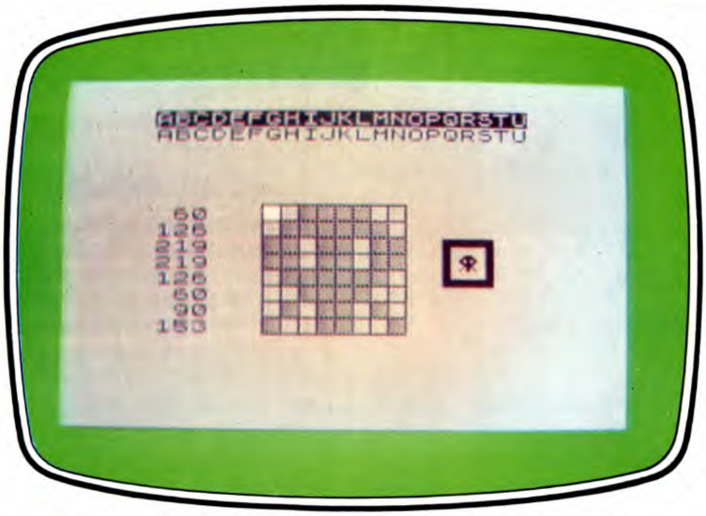
The UDG generator SAVEs them from the usual place in memory; if you are going to use more than one bank, you will need to use a *FX 2Ø call, and LOAD the extra blocks in above the old one. The article on pages 450 to 457 explains this.



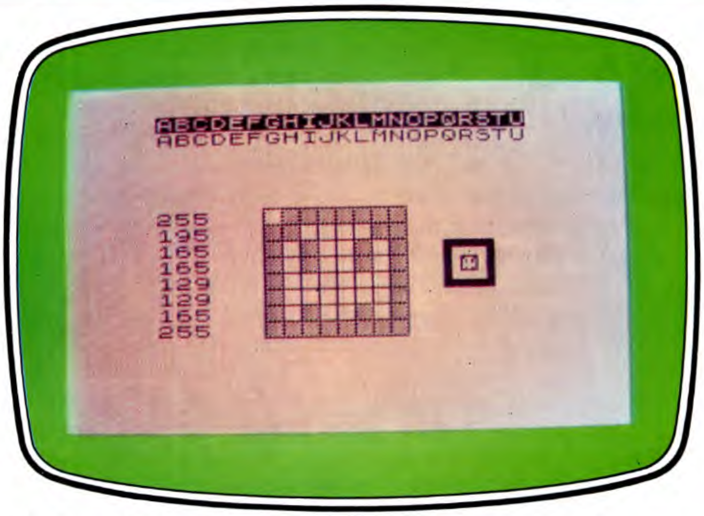
The last article gave you the first half of a program to make it easier for you to design your own graphics. This article includes the next half of the program to give you even more facilities to edit your UDGs. First, LOAD the last program, and then type in these extra lines.

If you have a Tandy, you should also make these changes to the new lines. Change the number 139 at the end of the DATA statement in Line 8Ø to 179, the number 48 to 237 and the number 8Ø56 to 8285; each of the numbers to be changed is in bold type to make them easier for you to find. In Lines 28ØØ and 29ØØ, change USRØ1 and USRØ2 to USR1 and USR2, respectively.

```
30 T=Ø:FORK=ØTO14:READN:
   T=T+N:POKEK+311ØØ,N:NEXT:
   READC:IFT< >C THENEND
4Ø T=Ø:FORK=ØTO84:READN:
   T=T+N:POKE3115Ø+K,N:NEXT:
   READC:IFT< >C THENEND
7Ø DATA 141,72,142,123,12,236,129,
   237,193,14Ø,123,84,38,247,57,1974
8Ø DATA 141,22,142,123,84,51,67,198,
   3,166,13Ø,167,194,9Ø,38,249,51,
   7Ø,14Ø,123,12,38,24Ø,57,189,139,48,52
9Ø DATA 6,31,3,142,123,14,134,3,183,
   121,68,23Ø,192,134,8,74,88,73,
   125,121,23,39,4,88,73,128,4,68
1ØØ DATA 1Ø2,132,125,121,23,39,3,68,
   1Ø2,132,77,38,23Ø,48,31,122,121,68
   38,219,48,6,14Ø,123,86,38,2Ø7,53,192,8Ø56
17Ø PRINT"□JOYSTICK OR KEYBOARD (J
   OR K) ?";
18Ø A$=INKEY$:IFA$< >"J"
   ANDA$< >"K" THEN18Ø
19Ø IF A$="J" THEN JY=1
35Ø IF JY=1 THEN GOSUB 1ØØØ ELSE
   GOSUB 15ØØ
1ØØØ PUT(X1,Y1)-(X1+5*T-1,
   Y1+4),C1,NOT
1Ø1Ø IF(PEEK(6528Ø)AND1)=Ø GOSUB2ØØØ
1Ø2Ø IFJOYSTK(1)=Ø THENY=Y-1
1Ø3Ø IFJOYSTK(1)=63 THENY=Y+1
1Ø4Ø IFJOYSTK(Ø)=Ø THENX=X-T
1Ø5Ø IFJOYSTK(Ø)=63 THEN X=X+T
1Ø6Ø RETURN
```



With the new routines for the *INPUT* UDG designer, you can turn your space invaders round to fly in four different directions



You can also INVERSE your design to the background colour. These UDGs were created on the Spectrum, but the other computers are similar

```
2100 GET(216,70)-(239,93),A
2110 GOSUB3000:GOTO2070
2200 A$=INKEY$:IFA$<>"S"
   ANDA$<>"P" THEN2200
2210 CLS:DN=0:IF A$="P" THEN
   DN=-2
2220 FORK=0TO14:FORR=0TO2
2230 PRINT#DN,PEEK(VARPTR
   (A(0))+K*3+R);:NEXT:PRINT #DN:NEXT
2240 IF DN<0 THEN2260
2250 A$=INKEY$:IFA$="" THEN2250
2260 FORK=15TO23:FORR=0TO2
2270 PRINT#DN,PEEK(VARPTR
   (A(0))+K*3+R);:NEXT:PRINT#DN:NEXT
2280 A$=INKEY$:IF A$="" AND DN=0
   THEN2280
2290 SCREEN1,ST:RETURN
2800 POKE30999,T-1:N=USR02
   (VARPTR(A(0)))
2810 GOSUB3000:GOTO2070
2900 POKE30999,T-1:N=USR01
   (VARPTR(A(0)))
2910 GOSUB3000:GOTO2070
```

Then add this to the end of Line 10:

```
:DEFUSR1=31100:DEFUSR2=31150
```

If you now RUN this new program, you can use the extra editing facilities. If the program stops before it does anything, check your DATA.

There are three main additional features: mirror, invert, and rotate a UDG.

To mirror a UDG you press the M key. The character becomes reversed from left to right—mirrored through an imaginary axis (a vertical line through the centre of the UDG). This is a very useful facility when you are defining two UDGs to stick together to make up one large character. If the large character is symmetrical, you need only define one character, store it, and then mirror it to get the second half.

The rotate facility is similar—pressing the R key rotates the whole UDG by 180 degrees. This can be useful if you need to have graphics which move both up and down—you can use the computer to produce an 'upside down' version of your UDG, for when the character is moving down.

The inverse facility, on the I key, actually reverses every pixel's colour. With PMODE 4 this is easy—green becomes black and vice versa, or buff becomes black, and vice versa.

On the other hand, when you are in colour mode (PMODE 3) things are not quite so simple. Blue and yellow are reversed (so that any blue pixel becomes yellow, and any yellow pixel becomes blue), as are red and green, buff and orange, and cyan and magenta.

Once you have got used to these colour changes, the facility is very useful. It is surprising how different some characters look when reversed like this.

This new program also allows you to change the screen colours while it is RUNning, by pressing the V key. What this does is to swap the set of screen colours you are using. If you were using buff and black, you can change to green and black (and vice versa) and if you were using either of the four-colour screens, using this command puts you into the other one.

Another new feature is the facility to print out the DATA values for the current bank of UDGs.

To do this you should press P. This prints out the set of numbers you should POKE into the top left-hand corner of the screen. Once you have POKEed them to the screen, you can then GET them into an array so that you can call them up to use as a UDG.

The option also lets you print the numbers to the screen in case you don't have a printer. You can then copy them down on paper, if you want a permanent record of them.

For this reason, after you have pressed the P key, the computer waits for you to press either P or S. P sends the printout to a printer, while S sends it to the screen.

USING YOUR UDGs

When you want to use the UDGs that you have designed in your own programs, you can CLOAD in the relevant data from tape, and then GET the bits you want into arrays using the command CLOADM.

Unfortunately, if you are going to use the UDGs in your own programs, you will either have to CLOADM the relevant pieces of data from tape, and then GET the arrays, every time you want to use the program or, alternatively, you need to transfer the information into data statements which the computer carries out whenever you RUN the program.

For this reason, it might be easier for you to either get a printout and then type in the numbers as data to be POKEd onto the screen, or for you to print the numbers to the screen (from the character generator program) and copy them down on paper. You can then incorporate them as DATA in your own program.

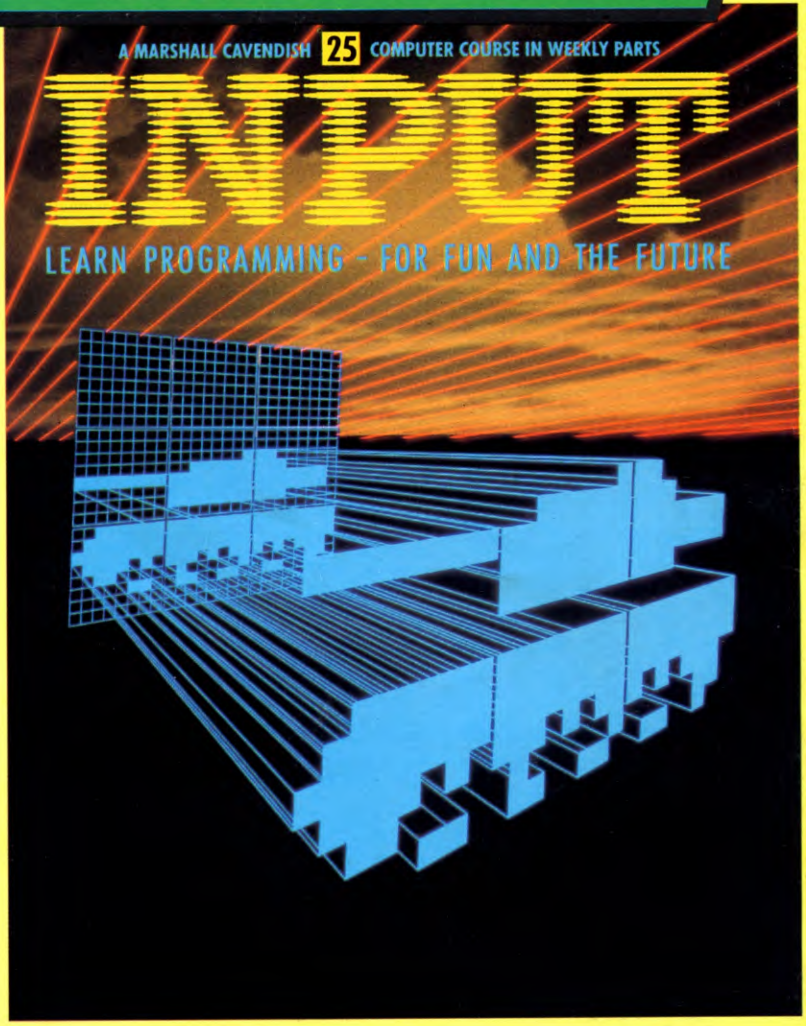
When you POKE the numbers onto the screen, you can then GET the area of the screen into an array for you to use as a UDG.

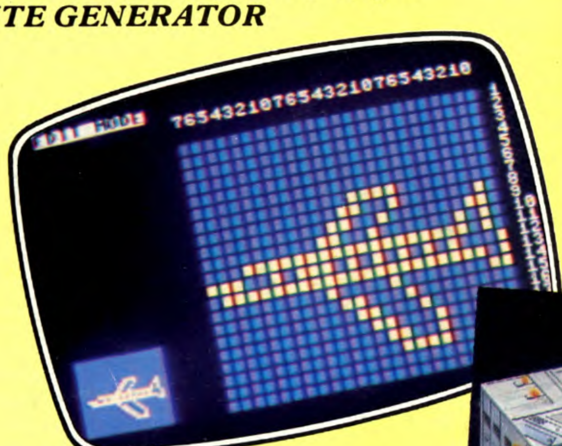
CUMULATIVE INDEX

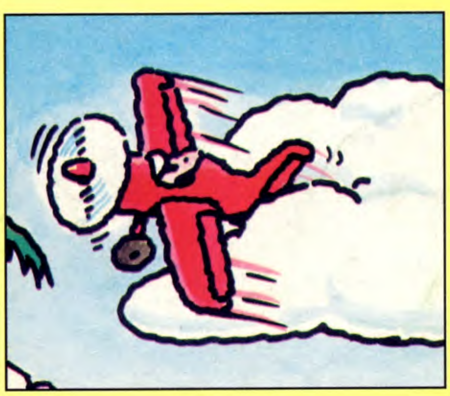
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

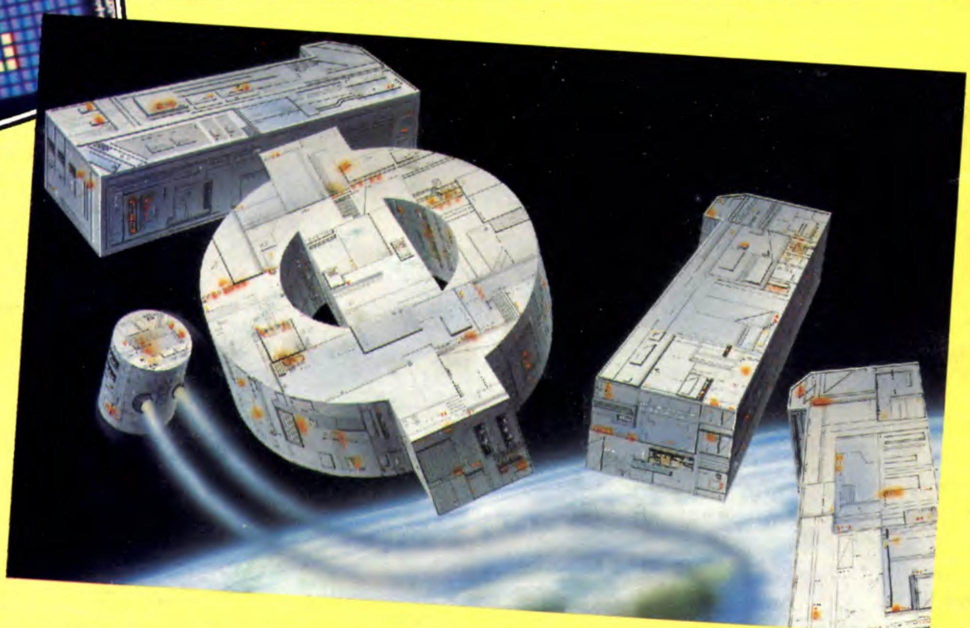
COMING IN ISSUE 25...

- Complete the FLIGHT SIMULATOR PROGRAM by adding the lines that allow you to take control of the aircraft
- If you wondered how the machine code frog and tank were created, there is a complete analysis of the FRAMEPRINT program used in INPUT 1 and 3
- If you want to get the most out of computer art, a GRAPHICS PAD is a powerful and versatile tool. Find out what one can do for you
- The way computers store numbers is more complicated than you might think, so there is a full guide to FLOATING POINT NUMBERS and EXPONENTS
- Plus, for COMMODORE users, a chance to get started on USING COLOUR SPRITES, with a useful SPRITE GENERATOR









ASK YOUR NEWSAGENT FOR INPUT